VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DISTRIBOUVANÝ SYSTÉM PRO INTERNETOVÉ MARKETINGOVÉ PRŮZKUMY

DISTRIBUTED IOS SYSTEM FOR INTERNET MARKETING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **RADOVAN KLEMBARA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN HRUBÝ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Klembara Radovan**
Program: Informační technologie
Název: **Distribuovaný systém pro internetové marketingové průzkumy**
Distributed iOS System for Internet Marketing
Kategorie: Elektronický obchod

Zadání:

1. Prostudujte programování aplikací pro iOS. Prostudujte programování serverových aplikací. Prostudujte současný stav zadávání elektronických dotazníků v prostředí Internetu.
2. Navrhněte systém složený ze serverové části a klientských stanic v podobě aplikací pro iOS. Systém umožní zadávat dotazníky, distribuovat dotazníky mezi registrované respondenty a vytvářet finální souhrny dotazníků. Navrhněte i vhodný motivační systém pro respondenty.
3. Systém složený ze serverové aplikace a mobilní aplikace v iOS implementujte.
4. Testujte systém v simulovaném provozu.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hrubý Martin, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Cieľom práce je navrhnúť a implementovať distribuovaný systém, ktorý bude zastrešovať prácu s internetovými dotazníkmi. Práca má za úlohu vytvoriť systém zložený z mobilnej aplikácie pre zariadenia s operačným systémom iOS a serverovej aplikácie. Úlohou mobilnej aplikácie je sprostredkúvať prácu s dotazníkmi. Serverová aplikácia sa postará o ukladanie a správu dát v databáze. Práca sa zameriava aj na vytvorenie motivačného systému, pre zlepšenie pravdepodobnosti vyplnenia dotazníkov.

V práci som vytvoril mobilnú aplikáciu pre zariadenia s iOS operačným systémom, ktorého verzia nie je staršia ako iOS 14. Ďalej som vytvoril serverovú aplikáciu, spravujúcu PostgreSQL databázu. V práci riešim aj motiváciu používateľov k vyplňovaniu dotazníkov, a to pomocou systému odmien. Používateľ dostáva za každý vyplnený dotazník určitú čiastku vnútoroaplikačnej meny. Menu si potom používatelia môžu vymeniť za zľavové kupóny.

Riešenie práce poskytuje používateľom mobilnú aplikáciu, v ktorej môžu efektívne vypĺňať dotazníky, získavať odmeny a vymieňať si ich za zľavové kupóny. Používateľom je umožnené vytvárať dotazníky a získavať z nich výsledky. Pre motivovanie ostatných používateľov je možné pridávať zľavové kupóny.

Abstract

The goal of this thesis is to design and implement distributed system, which covers work with internet surveys. The result of this thesis is a system consisting of mobile application for devices with iOS operating system and server application. Mobile application is created to mediate surveys operations. Server application maintains data in database. Thesis also focuses on creating the motivational system, to improve response rate.

In this thesis I created mobile application for devices with iOS operating system, which version is not older than iOS 14. I have created server application maintainig PostgreSQL database. In the thesis I am solving user motivation for filling surveys with a reward system. For each filled survey every user is rewarded by some amount of inner application currency. This currency can be exchanged for discount coupons.

Solution of this thesis brings mobile application to users, where they can efectively fill up surveys, get rewards and buy coupons. Users also can create surveys and are able to receive survey results. It is possible to add new coupons to motivate other users.

Kľúčové slová

Dotazníky, Mobilná aplikácia, Swift, Vapor, SwiftUI, Serverová aplikácia, Prieskumy, iOS, Apple, Motivácia, PostgreSQL, Combine, Zľavové kupóny

Keywords

Questionnaires, Mobile application, Swift, Vapor, SwiftUI, Server application, Surveys, iOS, Apple, Motivation, PostgreSQL, Combine, Discount coupons

Citácia

KLEMBARA, Radovan. *Distribouvaný systém pro internetové marketingové průzkumy.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Hrubý, Ph.D.

Distribouvaný systém pro internetové marketingové průzkumy

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne po vedením pána Ing. Martina Hrubého, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Radovan Klembara
4. mája 2021

Poďakovanie

Chcel by som poďakovať Ing. Martinovi Hrubému, Ph.D. za odbornú pomoc poskytnutú pri riešení tejto práce.

Obsah

1 Úvod **2**

2 Prieskum potrebných zdrojov a súčasných riešení **4**
2.1 Vytvorenie dotazníka 4
2.2 Popis súčasných riešení 7
2.3 Serverové aplikácie 9
2.4 Programovací jazyk Swift 10
2.4.1 Základy programovania v jazyku Swift 11
2.4.2 Programovanie aplikácií pre iOS zariadenia 13
2.4.3 Tvorba serverových aplikácií pomocou knižnice Vapor 18

3 Návrh jednotlivých častí systému **21**
3.1 Návrh životného cyklu dotazníku 21
3.2 Návrh mobilnej aplikácie 22
3.3 Návrh serverovej časti systému 25
3.4 Návrh motivačného systému 26

4 Implementácia **30**
4.1 Mobilná aplikácia 30
4.2 Serverová aplikácia 32
4.2.1 Autorizácia prístupu k zdrojom 33
4.2.2 Spracovanie požiadaviek 34

5 Testovanie **37**
5.1 Príprava pre testovanie 37
5.2 Testovanie pomocou simulovanej prevádzky systému 37
5.3 Závery vyvodené z výsledkov testovania 38

6 Záver **39**

Literatúra **40**

A Dotazník použitý pri testovaní **42**

B Obsah priloženého pamäťového média **43**

Kapitola 1

Úvod

Názor človeka na nejakú vec alebo jeho postoj k určitej problematike môže byť často veľmi dôležitou informáciou pre vedcov, firmy a iné subjekty. Pre získanie tejto informácie sa často používajú dotazníky. Zbieranie informácií pomocou dotazníkov má ale určité nevýhody. Jednou z hlavných nevýhod dotazníkov je problém ich distribúcie medzi jednotlivých respondentov alebo motivácia ľudí k ich vyplneniu.

Postupom času sa v rámci trendu informačnej doby prenáša všetko do elektronického sveta a dotazníky nie sú výnimkou. Aj keď je možné elektronické dotazníky distribuovať jednoduchšie a efektívnejšie, problémy s nimi spojené sa ale úplne nevyriešili.

V súčasnosti je elektronická forma dotazníkov často preferovaná pred ostatnými formami. Jednotlivé dotazníky sú potom prístupné pre respondentov často vo forme webových stránok. Respondentom je zvyčajne poslaná URL adresa webovej stránky, na ktorej je možné dotazník vyplniť. Tvorcovia dotazníkov preto musia mať nejaký zoznam respondentov, čo môže byť pre samotných tvorcov problematické alebo nepraktické. Z týchto dôvodov boli zriadené systémy, v ktorých sú dotazníky združované a respondenti sa do nich môžu prihlásiť. Prihlásený respondent potom má možnosť vybrať si dotazník, ktorý vyplní. Prihlásenie sa do systému respondentom je ale niečím navyše, k čomu musí byť respondent motivovaný. Najpoužívanejším motivačným prostriedkom je odmena, ktorá býva najčastejšie vo forme peňazí alebo zľavových kupónov.

Najpoužívanejšími takýmito systémami sa javia byť webové a mobilné aplikácie. Tvorcovia dotazníkov majú možnosť poslať dotazník správcom aplikácie, ktorí dotazník následne pridajú do systému. Respondenti sú schopní vypĺňať dotazníky pomocou webovej alebo mobilnej aplikácie. Za vyplnenie dotazníku správca zvyčajne posiela respondentovi odmenu. Existujúce riešenia sa ale postupne menej a menej zaoberajú samotnými dotazníkmi a zameriavajú sa na to, aby používateľ vykonával iné aktivity, za ktoré bude odmenený. Práve toto nové zameranie vytláča pôvodný zámer do pozadia, na čo doplácajú nie len tvorcovia dotazníkov, ale aj respondenti, ktorých hlavnou motiváciou je vyjadrenie svojho názoru.

Moja práca má za úlohu riešiť práve spomínané problémy a kompenzovať nevýhody elektronických dotazníkov. Pôjde o systém zložený z dvoch hlavných častí. Prvou časťou je mobilná aplikácia pre zariadenia pracujúce na operačnom systéme iOS. Aplikácia slúži ako hlavné médium pre prácu s dotazníkmi. Používateľ v nej môže prehľadávať a vypĺňovať dotazníky. Zároveň mu je sprístupnený aj jednoduchý spôsob ako vytvoriť a distribuovať dotazník. Za vyplnenie dotazníku dostane používateľ odmenu v podobe meny používanej v rámci aplikácie. Túto odmenu si môže v obchode, ktorý je súčasťou aplikácie, vymeniť za zľavové kupóny. Používateľ, ktorý chce prispieť k motivácii respondentov, môže pridať do obchodu nové kupóny. Pre uschovanie dôležitých informácií je tu druhá časť obsahujúca

serverovú aplikáciu a databázu. V databáze sú uložené dotazníky, údaje pre prihlasovanie používateľov a kupóny. Druhá časť sa ďalej stará o komunikáciu medzi aplikáciou a databázou. Ďalšou jej úlohou je zaobstarávanie práce s kupónmi.

Tvorba dotazníkov nie je úplne triviálnou záležitosťou, pretože správny dotazník má nejakú štruktúru a riadi sa určitými pravidlami. O tom, aké zásady je dobré dodržiavať pri tvorbe dotazníku sa pojednáva v nasledujúcej kapitole 2. V tejto kapitole sa taktiež popisuje, na čo si musí tvorca dotazníku pri tvorbe dávať pozor. Ďalej sa v nej hovorí o prípadnej motivácii respondentov. Mobilná aplikácia je vytvorená pre iOS zariadenia, a preto budú predstavené dôležité časti programovacieho jazyka Swift. V kapitole sú ďalej podrobne popísané existujúce riešenia, z ktorých som sa inšpiroval, poprípade poučil.

Produktom mojej práce je systém, ktorého veľkou súčasťou je grafické rozhranie. Podrobný popis tvorby rozhrania je spísaný v kapitole 3. Veľkou súčasťou tejto kapitoly je tiež návrh systému na základe poznámok z popisu už existujúcich riešení, ich predností a nedostatkov. Popisuje sa v nej aj návrh databázy, motivačného systému a komunikácie medzi jednotlivými účastníkmi systému.

Kapitola 4 popisuje zaujímavé alebo dôležité konštrukcie, ktoré boli použité pri implementovaní systému. Táto kapitola sa zameria na dve hlavné časti systému, a to mobilnú aplikáciu a serverovú aplikáciu. Mobilná aplikácia je napísaná v knižnici SwiftUI. Serverová časť je vytvorená pomocou knižnice Vapor. Serverová aj mobilná aplikácia ďalej používa aj knižnicu Combine. Ako databázový systém som zvolil PostgreSQL.

Každá časť musela byť testovaná, a preto bol systém testovaný simulovaním prevádzky. Súčasťou testovania bol aj krátky dotazník ohľadom spokojnosti. Bližší popis spôsobu a výsledkov testovania sa nachádza v kapitole 5. Súčasťou kapitoly je aj popis prípravy testovania.

V záverečnej kapitole 6 vyhodnocujem, či a ako dobre sa mi podarilo vytvoriť systém tak, ako bol zadaný a ako som si ho predstavoval. Súčasťou záveru je taktiež návrh na zlepšenie systému a popis jeho možného rozšírenia.

Kapitola 2

Prieskum potrebných zdrojov a súčasných riešení

S dotazníkmi sa stretávame často, či už pri predstavovaní nášho názoru na produkt alebo pri vyplňovaní rôznych žiadostí. Dotazník je sada otázok, na ktoré môže odpovedať skupina ľudí. *Respondent* je človek, ktorý zodpovedal na otázky z dotazníku. Tvorcu dotazníka je možné nazvať *bádateľom*, pretože sa za pomoci dotazníku snaží vybadať nejakú informáciu. Bádateľ pozbieraním odpovedí od respondentov a ich analyzovaním získava sadu dát. Zo získaných dát sa potom dajú vyvodiť určité závery.

Dotazníky sú pre štúdie veľmi výhodnou a často používanou formou získavania dát, pretože ako píše Naresh K. Malhotra vo svojej práci [8] dotazníky zabezpečujú štandardizáciu a porovnateľnosť bez ohľadu na to, kto zbieral dáta. Saul McLeod vo svojom článku [11] upozorňuje na to, že aby dáta z dotazníkov boli spoľahlivé, musia byť všetky otázky v rovnakom poradí u všetkých respondentov. Ďalej v ňom píše, že dotazníky zabezpečujú relatívne lacný, rýchly a efektívny spôsob získania veľkého množstva dát od veľkej vzorky ľudí. Ich výhodou, ako píše Stefan Debois vo svojom článku [3] je, že sa pomocou nich môže do výskumu pridať veľké množstvo respondentov, pričom následná analýza nemusí byť oveľa náročnejšia ako keby sa použil iný spôsob získavania dát. Zároveň je nutné pri analýze myslieť na to, že niektoré odpovede môžu byť skreslené.

Saul McLeod zároveň v článku [11] tvrdí, že respondenti často chcú vyzerať lepšie, a preto upravujú svoje odpovede. Takýto jav je možné sledovať pri respondentoch, ktorí vyplňujú otázky týkajúce sa napríklad množstva konzumovaného alkoholu, nikotínu alebo iných návykových látok. V knihe od Naresha K. Malhotru [10] sa píše, že jedným z nevýhod dotazníkov je aj fakt, že bádateľ sa musí spoliehať na sebakritickosť respondentov, ktorá ale nemusí byť vždy dostatočne pravdivá. Aby mal vytvorený dotazník zmysel, musí ho ale niekto aj vyplniť. Vyplnenie dotazníku si vyžaduje určité množstvo času, ktorý tým pádom musí respondent obetovať v prospech bádateľa. Respondent preto musí byť k tomuto motivovaný.

2.1 Vytvorenie dotazníka

Bádateľ (tvorca dotazníku) vytvára dotazník na základe potreby nejakých informácií. Počiatočnou fázou tvorby je upresnenie základnej otázky, na ktorú chce bádateľ získať odpoveď. Počiatočná otázka býva zväčša až príliš obsiahla. Stefan Debois píše v článku [3], že na príliš dlhé otázky respondenti nechcú odpovedať, a preto zvyknú neodpovedať na všetky

otázky. Navyše analýza odpovedí na prvotnú otázku je väčšinou až príliš náročná a celý proces by sa tým spomalil.

Keď je známa základná otázka, je vhodné zamerať sa na to, aké dáta chce bádateľ získať. Webová stránka od *QuestionPro Survey Software* [14] tvrdí, že typ požadovaných dát priamo určuje dizajn dotazníku. Napríklad pokiaľ je potrebné získať kvantitatívne nominálne dáta, Saul McLeod v článku [11] odporúča zvoliť uzatvorené otázky, ktorých možné očakávané odpovede sú kategorizované a priložené k otázke ako možnosti. Získanie iného typu dát si vyžaduje použitie iného typu otázok. Základná otázka je preto postupne rozdeľovaná na viacero menších otázok. Otázka je rozdeľovaná na menšie otázky dovtedy, kým jednotlivé otázky nie sú dostatočne jednoduché (žiadna otázka neobsahuje žiadnu podotázku). Otázky sú potom prípadne kategorizované a podľa logických celkov zoradené.

Typy otázok

Najpoužívanejšími typmi otázok sú otvorené, zatvorené alebo kombinované otázky. Na niektoré otázky sú vhodné komplikované odpovede a práve také odpovede získame pomocou otvorených otázok. Často je kladený dôraz na to, aby respondent uviedol rozsiahlejšiu odpoveď, v ktorej sa podrobnejšie vyjadrí k danej otázke. Takéto otázky získavajú podľa Saula McLeoda [11] takzvané kvalitatívne dáta. Výhodou je detailnejšia odpoveď. Takáto odpoveď lepšie zobrazuje postoj respondenta a vysvetľuje, prečo zaujal taký postoj, aký zaujal. Nevýhodou je, že analýza odpovedí na takéto otázky je zložitá a zdĺhavá. Otázky tohoto typu podľa Saul McLeoda [11] nie sú vhodné pre menej vzdelaných respondentov, pretože vyžadujú určitú schopnosť vyjadrovania sa.

Ak je potrebné získať kvantifikované dáta, jasnou voľbou sú uzavreté otázky. Sú to otázky, ktoré už majú preddefinované očakávané odpovede. Otázky tohoto typu hovoria o tom, koľko respondentov odpovedalo na otázku určitou odpoveďou. Čo sa týka pomeru cena za čas strávený vypĺňaním dotazníka alebo analýzou získaných dát, sú najvýhodnejšie. Analýza nominálnych dát potom spočíva v jednoduchom vytvorení štatistík z početnosti zodpovedaných hodnôt [11]. Nevýhodou takéhoto prístupu je, ako poznamenal Saul McLeod v článku [11] aj to, že odpovede nemajú práve najlepšiu výpovednú hodnotu.

Špeciálnymi typmi zatvorených otázok sú napríklad aj obrázkové alebo stupnicové otázky. Odpovede na tzv. obrázkové otázky sú obrázkami, z ktorých respondent vyberá. Respondenti takéto zatvorené otázky riešia spravidla rýchlejšie ako iné typy otázok. Dôvodom je, že obrázkovým odpovediam dokáže respondent rýchlejšie a lepšie porozumieť, a tak je schopný rýchlejšie vybrať jednu z možností. Obľúbeným typom uzatvorenej otázky je aj stupnicová otázka, ktorá vzniká vymenovaním odpovedí, ktoré tvoria stupnicu.

Stupnicové otázky majú za úlohu zistiť stupeň súhlasu, respektíve nesúhlasu, respondenta s nejakým tvrdením. Väčšinou je stupnica vyvážená, a teda má rovnaký počet negatívnych ako aj pozitívnych odpovedí. Stupnica máva skoro vždy jednu neutrálnu odpoveď, aby sa zabránilo skresleným dátam. Najpoužívanejšou takouto stupnicou je podľa Naresha K. Malhotru [8] *Likertova stupnica*, ktorá má päť stupňov: veľmi nesúhlasím, nesúhlasím, ani súhlasím ani nesúhlasím, súhlasím a veľmi súhlasím.

Kompromisom medzi otvorenými a zatvorenými otázkami sú otázky kombinované. Odpovede kombinovaných otázok sú vymenované, ale nachádza sa tu navyše jedna odpoveď umožňujúca respondentovi odpovedať otvorene. Tieto otázky umožňujú zistiť, či očakávané odpovede boli správne alebo či respondenti mali skôr tendenciu odpovedať inak, ako bolo predpokladané. Analýza dát je pomalšia ale aj napriek tomu presnejšia ako pri zatvorených otázkach. Kombinované otázky zároveň redukujú zavádzajúce dáta získané od responden-

tov, ktorí si neboli istí svojou odpoveďou alebo len nevedeli, ktorú z odpovedí majú vybrať v ich konkrétnom prípade.

Dizajn dotazníkov

Dizajn dotazníku má veľký vplyv na to, či respondent vyplní celý dotazník alebo či sa ho rozhodne zanechať nevyplnený. Formulácia otázok podľa článku Sofie Nelen [12] musí byť jednoduchá a nesmie byť smerodajná. Slovná zásoba podľa práce Naresha K. Malhotru [8] nesmie byť neadekvátna k jazykovým schopnostiam dopytovanej skupiny. Saul McLeod [11] upozorňuje aj na potrebu dať pozor na technický žargón, ktorému nemusia všetci rozumieť. Otázky by podľa Naresha K. Malhotru [8] mal dizajnér skontrolovať vždy štyrmi dotazmi, a tak získať povedomie o tom ako ich budú vnímať respondenti. Týmito dotazmi sú: „Kto?“, „Kedy?“, „Kde?“ a „Čo?“.

Dotazníky podľa Sofie Nelen [12] by mali byť čo najkratšie, či už v počte otázok, tak v rozsahu ich znení a ich možných odpovediach, pretože týmto sa riskuje, že respondent ukončí vyplňovanie v polovici dotazníku. Rozsiahle dotazníky môžu respondentov unaviť. To spôsobí, že respondent odpovedá skreslene a buduje si negatívny postoj k dotazníkom ako takým. Je dobré po spísaní otázok sa nad všetkými zamyslieť a utvrdiť sa v tom, či ich bádateľ potrebuje a či sú všetky k téme. Otázky by mali podľa Naresha K. Malhotru [8] získavať iba potrebné dáta a nie dáta, ktoré sú pre nás zaujímavé. Ďalšou dôležitou súčasťou je zoradenie otázok. Zoradenie by malo odpovedať logickému postupu a obzvlášť by mali byť osobné otázky až na konci dotazníka.

Respondent si počas vyplňovania buduje určitý vzťah k danému dotazníku. Na základe pozitívneho vzťahu je potom menej pravdepodobné, že ho respondent nedokončí kvôli príliš osobným otázkam. Odpovede na intímne otázky by mali byť kategorizované tak, aby si respondent mohol vybrať, do ktorej skupiny patrí.

Otvoriť dotazník jednoduchými a neutrálnymi otázkami podporuje respondentov k jeho vyplneniu, a tak napomáha s budovaním spomínaného pozitívneho vzťahu [8]. Otvorené otázky na začiatku dotazníka sú tak isto veľmi dobré, pretože umožňujú bádateľovi získať povedomie o tom, aký veľký prehľad má respondent v danej téme. Taká informácia je podstatná pri vyhodnocovaní výsledkov. Dobrou praxou je aj pridanie neutrálnej odpovedi, ktorú respondent môže označiť, pokiaľ si nie je istý odpoveďou alebo odpoveď nepozná. Takáto odpoveď znateľne redukuje skreslené dáta. Po vytvorení dotazníka je dobré ho otestovať na 15-30 respondentoch, a tak zistiť, či sú otázky správne pochopené. Zistí sa taktiež aj čas potrebný na vyplnenie dotazníka.

Motivácia respondentov

Podľa článku zo žurnálu Journal of Official Statistics [15] sa ukazuje, že počet respondentov každým rokom vo väčšine štátov klesá. Aby sa tento trend zmenil, je nutné motivovať ľudí k ich vyplňovaniu. Podľa článku od Nigela Lindemanna [9] 38% ľudí je motivovaných vyplniť dotazníky na základe túžby vyjadriť svoj názor, a tým zapadnúť do nejakej skupiny ľudí. Dotazníky môžeme preto upraviť tak aby boli jednoduché na vyplnenie a aby boli ľahko čitateľné. Výsledky, ku ktorým sa bádateľ dopracuje, by mali byť zverejnené tak, aby sa zachovala anonymita respondentov. Vhodný prístup je podľa Nigela Lindemanna [9] aj poslanie predbežných oznámení. Takéto oznámenie by malo byť personalizované napríklad pozdravom, v ktorom vystupuje meno respondenta. Oznámenie by malo obsahovať všeobecné informácie popisujúce daný dotazník, ako napríklad na čo sa použijú výsledky alebo kto za ním stojí. Podľa článku od Victorie Duff [4] potencionálni respondenti zvyknú zabud-

núť na dotazník, a preto je dobré im ho pripomenúť, ale toto musí byť spravené s mierou. Poďakovanie za vyplnenie dotazníku je tak isto dobrou praktikou. Najlepšou motiváciou však zostáva materiálna alebo elektronická odmena za vyplnenie dotazníku. Takouto odmenou najčastejšie býva nejaká suma peňazí prípadne zľavové kupóny.

2.2 Popis súčasných riešení

V súčasnosti sa problematika dotazníkov zvyčajne rieši rozdelením systému na časti. Distribúciu dotazníkov má na starosti nejaká mobilná aplikácia, do ktorej správcovia pridávajú dotazníky na základe žiadostí bádateľov. Používateľovi ale nestačí, aby si stiahol a nainštaloval danú aplikáciu, musí sa do nej aj registrovať, aby správca systému mohol zaručiť, že jeden respondent nevyplní ten istý dotazník viac krát. Bádateľ môže dotazník vytvoriť napríklad vo webovej aplikácii. Používateľ si v mobilnej aplikácii môže nájsť dotazník, ktorý chce vyplniť. Avšak samotné vyplnenie dotazníku v rámci mobilnej aplikácie nebýva podporované. Aplikácia používateľa presmerováva väčšinou do webovej aplikácie správcu aplikácie alebo bádateľa. Presmerovaný používateľ sa musí častokrát znovu registrovať do tejto aplikácie, aby sa dostal k danému dotazníku. Po vyplnení dotazníka sú používateľovi prezentované ďalšie dotazníky bádateľa alebo je presmerovaný naspäť do mobilnej aplikácie.

Motivácia respondentov stojí na princípe odmien. Za každý správne vyplnený dotazník dostane respondent odmenu v podobe nejakej meny. Používaná mena nemusí byť len obecnou ako sú eurá alebo doláre, ale môže byť len akousi abstrakciou, ktorá je v nasledujúcom texte pomenovaná ako tokeny. Takáto štruktúra systému si potom vyžaduje určitú kontrolu identity používateľa, nakoľko je nutné nejako zaistiť, aby používateľ, ktorý správne vyplnil dotazník, dostal odmenu. Túto kontrolu zaisťuje komunikácia medzi bádateľom a správcom aplikácie, pre automatizáciu celého procesu sa často používa rozhranie REST. Pre zlepšenie motivácie býva súčasťou aplikácie nejaký prvok, ktorý prenáša určitú čiastku získanej odmeny na fyzicky dostupnú formu. Takýmto prvkom býva obchod, v ktorom si používateľ dokáže vymeniť odmenu za zľavové kupóny, alebo si môže nechať previesť peňažný obnos na jeho bankový účet.

Make Money!

Jedná sa o iOS a android aplikáciu určenú na získavanie tokenov. Okrem vypĺňania dotazníkov aplikácia umožňuje získavať tokeny aj skúšaním rôznych aplikácií. Podmienky pre získanie odmeny sú vždy presne definované a ich splnením používateľ automaticky získava sľúbený obnos tokenov. Určitú čiastku tokenov získava používateľ aj denne. Po dosiahnutí dostatku tokenov je možné zameniť tieto tokeny za doláre, ktoré majú byť odoslané pomocou služby PayPal používateľovi na účet. Poskytované dotazníky k vyplneniu nie sú súčasťou aplikácie a pre ich vyplnenie je používateľ presmerovaný do webového prehliadača na webové stránky bádateľa. Presmerovaný používateľ sa väčšinou musí pred vyplnením dotazníka registrovať na týchto stránkach. Niektoré takéto stránky sú chybné, a tak nemusí byť dotazník používateľovi sprístupnený.

CashCamel

Android aj iOS aplikácia zameraná priamo na dotazníky. Jednotlivé dotazníky môžu byť vyplnené v aplikácii, ale častejšie sa objavujú dotazníky, ktoré presmerovávajú používateľov na webové stránky zmluvných partnerov aplikácie. Používateľ za každý dotazník získava pe-

ňažnú čiastku v dolároch. Keď používateľ nadobudne čiastku väčšiu ako 10 dolárov, môže si 10 dolárov nechať vyplatiť, a to na účet pomocou služby PayPal. Na hlavnej stránke pribúdajú nové dotazníky približne jedenkrát za dva týždne. Väčšina dotazníkov má určité neznáme špecifikácie a po vyplnení dotazníku, o ktorom používateľ nevie či je tým, ktorý chcel vyplňovať, je v mnohých prípadoch používateľovi oznámené, že sa nehodí pre daný prieskum, a teda sa nemôže prieskumu zúčastniť. Takéto prípady sú bežné hlavne pri dotazníkoch, ku ktorým sa používateľ dostane po presmerovaní. Pokiaľ ale vyplnenie prebehne bez problémov, je odmena automaticky zaslaná používateľovi.

Money Machine

Aplikácia určená pre iOS zariadenia, ktorá sa zameriava okrem dotazníkov aj na propagovanie aplikácií. Používateľ získava tokeny za vypĺňanie dotazníkov a za skúšanie aplikácií. Po získaní dostatku tokenov si ich používateľ môže zameniť za doláre, ktoré mu môžu byť poslané pomocou PayPal služby, bankovým prevodom alebo si ich ešte môže zameniť za poukážku do obchodov partnerov aplikácie (Amazon, Zalando, Digitec). Aby používateľ získal odmenu, často musí po vyplnení dotazníku splniť ďalšie dodatočné požiadavky (hlavne stiahnutie nejakej aplikácie a následná registrácia do systému tejto aplikácie). Po splnení požiadaviek ale používateľ musí ešte čakať na správcu aplikácie, aby potvrdil splnenie požiadaviek. Potvrdenie by nemalo podľa oficiálnych informácií aplikácie[1] trvať dlhšie ako 24 hodín. Dizajn aplikácie sa javí ako zastaralý a dotazníky spolu s inými možnosťami pre získanie tokenov nie sú pridávané do aplikácie často.

ySense

Mobilná aplikácia, v ktorej môže bádateľ pozývať jednotlivých používateľov k vyplneniu dotazníkov. Pokiaľ používateľ nie je pozvaný k vyplňovaniu, musí si nejaký dotazník nájsť medzi ponukami. Ponuky sú ale neprehľadné a schované v rôznych článkoch v sekcii *blog*. Mnoho ponúk vyžaduje ďalšiu registráciu a sťahovanie ďalších aplikácií. Po vyplnení dotazníku, u ktorého je ťažké rozoznať, či sa jedná o vybraný dotazník používateľom, môže byť respondentovi oznámené, že sa nehodí na prieskum, a teda nemôže vyplniť pôvodný dotazník. Používateľ získava za vypĺňanie dotazníkov doláre, ktoré si potom môže poslať v podobe poukážky (Amazon, Payoneer, Skrill, Reward Link Italy) alebo si ich môže nechať zaslať pomocou služby PayPal na účet. Veľkou súčasťou systému tvorí fakt, že používateľ musí byť pozvaný na vyplnenie väčšiny dotazníkov.

TGMPanel

Webová aplikácia zameraná na dotazníky. Vypĺňaním dotazníkov používateľ získava body, ktoré potom môže vymeniť za eurá a tie si môže nechať poslať na účet pomocou služby PayPal. Avšak takto môže urobiť len ak má dostatočný počet bodov. Body získa aj po vyplnení dotazníkov v osobnom profile. Na hlavnej stránke sa zriedkavo objavujú dotazníky, ktoré ale sú na iných webových stránkach a po ich vyplnení je často používateľovi oznámené, že sa nehodí na prieskum. Aplikácia je ale stále v štádiu vývoja, a tak sa používateľ môže stretnúť s nejakými chybami či nekonzistentnými dátami.

[1] https://apps.apple.com/rs/app/moneymachine-make-money/id1512778733

Survey Monkey

Jedná sa o najrozšírenejší systém pre tvorbu dotazníkov. Distribúcia je riešená pomocou posielania adresy webovej stránky respondentom. Dotazníky je možné vypĺňať aj v rámci aplikácie, ale to si vyžaduje použitie tzv. *Kiosk* módu. V tomto móde je ale povolené vyplňovanie iba na zariadení, na ktorom je bádateľ prihlásený do svojho účtu. Používateľ preto zväčša vypĺňa dotazníky v prehliadači, kde mu webová aplikácia prezentuje jednotlivé otázky. Súčasťou systému nie je žiadna hlavná stránka zobrazujúca jednotlivé dotazníky. Jediným spôsobom, ako sa užívateľ dostane k dotazníku je, že mu je zaslaná URL adresa dotazníka. Vytvoriť dotazník môže ktorýkoľvek registrovaný používateľ. Pri tvorbe sú mu poskytnuté bežné prvky dotazníku, ale pokiaľ sa prihlási k platenému prístupu, získa oveľa viac možností. Platiaci používateľ si môže pridať k dotazníku logo alebo takýto používateľ nemá žiadny limit čo sa týka počtu otázok v dotazníku.

2.3 Serverové aplikácie

Serverová aplikácia má za úlohu sprostredkovávať zdroje jednému alebo častejšie viacerým používateľom. Pričom medzi typmi štruktúr počítačových sietí je možné nájsť napríklad štruktúru klient-server, ktorá naznačuje prácu serverovej aplikácie. Ide teda o štruktúru, v ktorej sa nachádza jedno hlavné zariadenie (server), ktoré je zapojené do siete a má k dispozícii nejaké zdroje. Do siete sú zapojené aj ďalšie zariadenia, ktoré chcú mať prístup k zdrojom servera. Tieto zariadenia sa označujú ako klientské stanice alebo ako klienti. Serverová aplikácia čaká na požiadavky od klientov. Požiadavky pre server sa líšia podľa toho, čo klienti vyžadujú. Požiadavky môžu byť posielané napríklad pomocou protokolu HTTP.

Protokol HTTP

Skratka HTTP pochádza z anglického názvu protokolu (Hypertext Transfer Protocol alebo v slovenčine Hypertextový Prenosový Protokol) používaného na celosvetovej sieti (World Wide Web v skratke WWW). Pracuje nad aplikačnou vrstvou popísanou modelom vzájomného prepojenia otvorených systémov (používa sa skratka OSI z anglického prekladu The Open Systems Interconnection) [7]. Protokol pracuje nad transportným protokolom kontrolovaného prenosu (známa je skratka TCP z anglického výrazu The Transmission Control Protocol).

Rozličné typy HTTP požiadavok majú rozličné sémantiky, ktoré ale nemusia byť dostatočne detailné pre komplexnejšie servery. Preto servery zvyknú zhlukovať prácu nad nejakými dátami. K týmto zhlukom aktivít nad dátami sa potom pristupuje pomocou jednotného identifikátora zdroja (Uniform Resource Identifier alebo častejšie používaná skratka URI), ktorý je súčasťou URL v HTTP požiadavke. HTTP protokol používa deväť typov požiadaviek, ktoré sú pomenované ako metódy. Najpoužívanejšími, a teda najčastejšie podporovanými metódami sú GET, POST, PUT a DELETE. GET je metóda používaná k získaniu nejakých dát. Metóda POST má za úlohu predať dáta serveru. PUT sa používa pre nahradenie nejakých dát. Požiadavka DELETE je určená k zmazaniu dokumentu.

Každá požiadavka musí mať hlavnú hlavičku. V hlavnej hlavičke sú vymenované ďalšie hlavičky a ich hodnoty. Hlavičkami sú napríklad dĺžka obsahu tela (Content-length) alebo hlavička oprávneného prístupu (Authorization), používaná pre predanie autorizač-

ného kódu. Za hlavičkou môže nasledovať telo, v ktorom sa môžu nachádzať akékoľvek zakódované dáta.

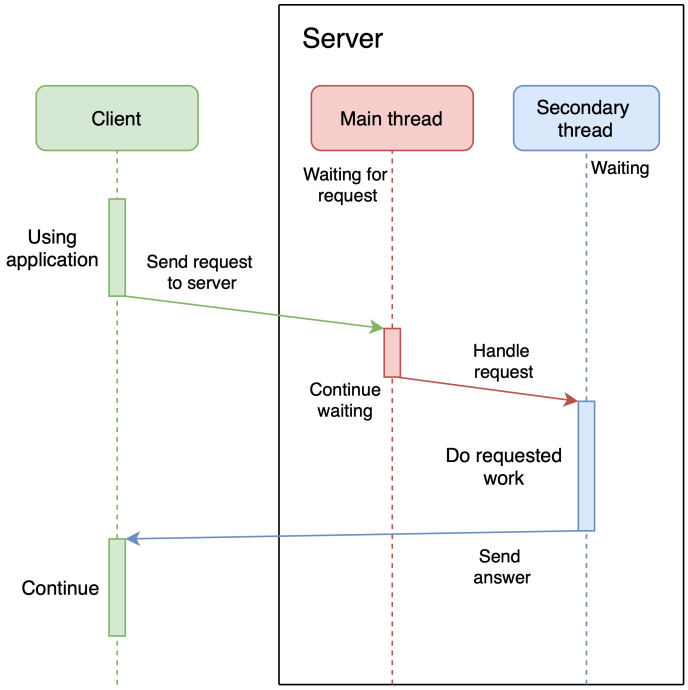
Programovanie serverových aplikácií

Pri programovaní serverovej aplikácie je nutné najskôr konfigurovať server. Táto konfigurácia spočíva v tom, že sa vytvorí a nastaví akýsi koncový bod (v angličtine využívaný termín socket), ktorý spája bránu (ďalej sa používa iba anglický ekvivalent - port) zariadenia s aplikáciou. Touto konfiguráciou sa dáva najavo operačnému systému, aby dáta poslané tomuto zariadeniu na tento port prepúšťal aplikácii. Ďalším krokom zo strany servera je počúvanie na danom porte. Počúvaním sa myslí čakanie na to, kým operačný systém nerozozná, že prišla požiadavka pre aplikáciu na určený port. Operačný systém prepošle požiadavku aplikácii. Požiadavka by potom mala byť serverom skontrolovaná a mala by sa vykonať nejaká práca. Často vyžadovanou praktikou je, aby serverová aplikácia vždy odpovedala klientovi správou, v ktorej je minimálne informácia o tom, či bola požiadavka správne spracovaná.

Takáto jednoduchá aplikácia bude síce fungovať korektne za určitých okolností, no dnešný trend ju považuje za neschopnú k bežnému fungovaniu. Pokiaľ príde nová požiadavka v čase, kedy sa spracováva predošlá, je táto požiadavka ignorovaná. Aby sa predišlo zahodeniu správnej požiadavky kvôli vyťaženiu, môžu byť požiadavky ukladané do poľa a postupne spracovávané. Uložené požiadavky sa ale nemusia dostať k spracovaniu dostatočné rýchlo, a tak im môže vyprchať čas. Preto sa využívajú procesy a, alebo vlákna. Aplikácia potom má jeden hlavný proces, ktorý v nekonečnom cykle čaká na požiadavky. Po prijatí požiadavky hlavný proces vytvorí nové vlákno, ktoré sa postará o požiadavku a potom pokračuje v čakaní. Vedľajšie vlákno spracuje požiadavku, vykoná nejaké operácie a odošle odpoveď späť klientovi. Obrázok 2.1 zobrazuje takéto paralelné prijatie požiadavky. Pre jednotlivé podporované typy požiadaviek a ich možné varianty či podmnožiny potom programátor vytvára cesty (routes). Cesty majú identifikátor a nejakú funkcionalitu. Vďaka Identifikátoru je možné ďalej rozdeľovať zdroje serverovej aplikácie.

2.4 Programovací jazyk Swift

Swift je silne typovaný, kompilovaný moderný jazyk podporujúci moderné trendy. Prvý krát bol predstavený v roku 2014 na konferencii Apple Worldwide Developers Conference (WWDC). Spoločnosť Apple Inc. vyvinula jazyk Swift ako moderný jazyk, ktorý mal nahradiť Objective-C na ich platformách. Podľa knihy od Johna Hoffmana [5] ponúka koncepty ako typový záver (type inference), generickosť, uzáverovú syntax, voliteľné typy a mnoho ďalších. Swift sa vďaka týmto konceptom stal krátko po predstavení jedným z najrýchlejšie rastúcich jazykov. Prispel k tomu aj fakt, že v roku 2015 spoločnosť Apple sprístupnila zdrojový kód širokej verejnosti, čím sa celý Swift stal projektom otvoreného kódu (open-source project). Ďalším zásadným vplyvom bolo to, že i keď bol Swift určený pre Apple platformy, je možné jeho kód skompilovať s obmedzeniami aj na iných platformách ako Linux alebo Windows. Obmedzenia pri kompilácii na iných platformách sa ale rýchlo zmenšujú a postupne zväčšujúca sa podpora Swiftu operačným systémom Ubuntu dovoľuje vývoj serverových aplikácií aj v jazyku Swift. Podľa oficiálnej stránky [2] je jazyk Swift intuitívny a dizajnovaný pre bezpečnosť aplikácií.



Obr. 2.1: Paralelizmus pri spracovaní požiadavok serverom.

2.4.1 Základy programovania v jazyku Swift

Jazyk Swift sa odvíja od jazykov inšpirovaných jazykom C. Aj keď bol navrhnutý ako náhrada za Objective-C, je možné ho s ním kombinovať. Navyše je možná kombinácia aj s jazykmi C++ a C vďaka jeho kompilátoru LLVM[2]. Aby sa uvoľnila operačná pamäť, Swift využíva pre prácu s pamäťou koncept deterministického držania informácie o počte referencií (reference counting). Vďaka tomuto konceptu nie je potrebný zberač odpadu (garbage collector), vyhľadávajúci nedosiahnuteľnú pamäť. Počet referencií sa navyšuje pri vytvorení silno odkazujúcich premenných. Pri zrušení odkazovania sa na objekt tento počet klesne. Vždy keď je počet referencií znížený na nulu, je pamäť automaticky uvoľnená.

Swift sa zbavuje aj nebezpečných konceptov ako sú premenné, ktoré neboli inicializované. Ďalej pomáha zabrániť nesprávnej práci s premennými, ktoré môžu nadobudnúť hodnotu `nil` (reprezentujúcu `NULL`). Typ takýchto premenných je označený znakom `?` (otáznik) a reprezentujú tzv. voliteľnú (optional) hodnotu. Jazyk Swift ponúka špeciálne konštrukcie pre prácu s voliteľným typom, ktoré donucujú programátorov, aby explicitne riešili prípad, kedy premenná nadobudne hodnotu `nil`. Jednou z týchto konštrukcií je aj `!` (výkričník). Jeho úlohou je pristúpiť na hodnotu voliteľnej premennej. Jeho použitie je ale nebezpečné, pretože pokiaľ sa v premennej nenachádza žiadna hodnota, respektíve je táto hodnota `nil`, celá aplikácia spadne. Výpis 2.1 názorne ukazuje ako sa voliteľný typ správa a ako je možné s ním pracovať.

```
1 // Definition of constatnt with optional type Int and value 2
2 let optionalValue: Int? = 2// type Int? == Optional<Int>
3 // Error optional value must be unwrapped
4 let onePlusOptionalValueErr = 1+optionalValue
```

[2] https://llvm.org

```swift
// If optionalValue is nil whole application shuts doum
let onePlusOptionalValueBreak = 1+optionalValue!
// If optionalValue is nil add zero
let onePlusOptionalValueOrDefaultValueZero = 1+(optionalValue ?? 0)
// If optionalValue is not nil copy its value into constatnt notOptional
if let notOptional = optionalValue {
    let onePlusOptionalValue = 1+notOptional
    // Do work with onePlusOptionalValue
} else {
    // Handle nil value
}
```

Výpis 2.1: Demonštrácia prístupu k hodnote premennej typu voliteľny `Int`.

Vymenovací typ Enum

Rozdielom medzi bežne podporovanými typmi vymenovania a typom `Enum` vo Swifte je, že Swift podporuje naviazanie hodnoty. To znamená, že ku každému prípadu vymenovania je možné priradiť hodnotu. Priradenie hodnôt je možné dvoma spôsobmi. Prvý spôsob určuje typ hodnôt všetkých prípadov. V rámci daného typu je potom možné každému prípadu vymenovania priradiť nejakú konštantnú surovú hodnotu (raw value).

Druhou možnosťou je typ vymenovania s priradenými hodnotami (associated values). Dovoľuje programátorovi ku každému prípadu definovať typ hodnoty, ktorá môže byť uložená do tohoto prípadu vymenovania. Deklarácia takéhoto typu je vo Výpise 2.2.

```swift
// Declaration of Enum type with associated values
enum EnumExampleRawVal {
    // All cases can have different types of associated values
    case First(String)
    // Cases can have different number of parameters
    case Second(Int, String)
    case Third([String])
}
```

Výpis 2.2: Deklarácia vymenovacieho typu s priradenými hodnotami.

Protokol v jazyku Swift

Swift podporuje aj protokolové programovanie. To je podobné s vyžadovaným prepísaním (override/overload) dedičnosti v objektoch. Ide o koncept, v ktorom existuje protokol, respektíve plán popisujúci a vyžadujúci určité metódy alebo vlastnosti (properties). Potom tento plán môže byť prijatý (adopted) štruktúrami, triedami alebo typom vymenovania. Prijatie spočíva v implementovaní všetkých požiadaviek protokolu.

Dátový typ štruktúra a trieda

Štruktúra je vo Swifte oproti iným jazykom rozšírená tak, že je možné v nej definovať metódy, čím sa stáva na prvý pohľad nerozlíšiteľnou od triedy. Rozdielom je, ako sa s nimi pracuje v rámci operačnej pamäti. Štruktúra sa predáva hodnotou. To znamená, že pri predaní štruktúry do funkcie sa skopíruje celá inštancia štruktúry na zásobník. Narozdiel

od štruktúry je trieda predávaná odkazom, z čoho vyplýva, že pri predaní do funkcie je skopírovaný iba odkaz na miesto v pamäti, kde sa inštancia triedy nachádza.

Uzávery

Nemenované bloky kódu, ktoré je možné vykonať v inej časti kódu, v akej boli definované sa vo Swifte nazývajú uzávermi (closures). Swift sa od iných programovacích jazykov, ktoré ponúkajú podobné koncepty líši tým, že uzávery si okrem daného kódu ukladajú aj ich kontext. Swift ponúka veľkú syntaktickú podporu pre uzávery. Dovoľuje napríklad písať uzáver za volanie funkcie, ktorá dostane tento uzáver ako posledný parameter.

Reaktívne programovanie v jazyku Swift

Reaktívne programovanie hovorí o spracovaní informácií v *prúdoch*. Prúdom sa myslia nejaké dáta, ktoré sú z niekade odoslané a niekam smerujú, pričom súčasťou prúdu je aj niečo, čím je možné propagovať zmenu stavu dát. Toto si vyžaduje troch *účastníkov* procesu. Prvým je odberateľ (subscriber), ktorý je akýmsi koncovým bodom. Jeho úlohou je posielanie požiadaviek ďalšiemu účastníkovi. Tieto požiadavky obsahujú informácie ohľadom množstva dát, ktoré odberateľ vyžaduje. Požiadavky sú zasielané vydavateľovi (publisher), ktorý zasiela dáta odberateľovi. Posledným účastníkom je operátor (operator), ktorý spája odberateľa a vydavateľa. Úlohou operátora je tiež kontrola a transformácia posielaných dát. Pre reaktívne programovanie pripravila spoločnosť Apple Inc. aj knižnicu *Combine*.

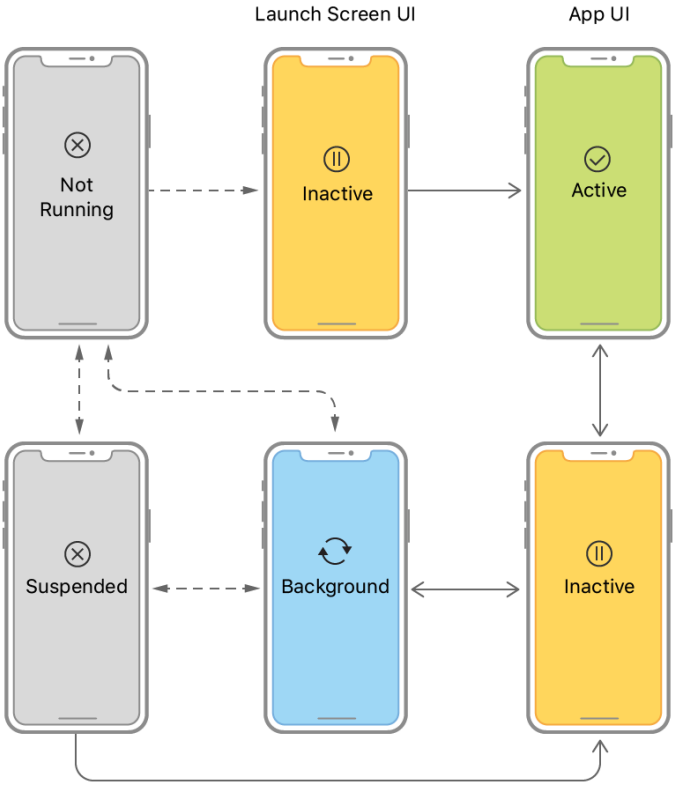
2.4.2 Programovanie aplikácií pre iOS zariadenia

Pre programovanie mobilných aplikácií, si musí programátor uvedomiť, že aplikácia sa musí stále o niečo starať. To znamená, že musí byť vždy pripravená na rôzne udalosti, ako napríklad minimalizácia aplikácie, či už používateľom alebo prichádzajúcim telefonátom. Apple preto presne definuje stavy, v akých sa aplikácia môže nachádzať. Každý zo stavov má aj určené, čo aplikácia môže v tomto stave robiť a čo nie. Stavom sa označuje to, či sú okná aplikácie na popredí, na pozadí alebo či prechádzajú do jedného z týchto stavov.

Životný cyklus aplikácie

Ako aplikácia pracuje a mení svoj stav, programátor musí upraviť jej správanie aby vyhovovalo aktuálnemu stavu. Napríklad ak je aplikácia na pozadí, nemala by vykonávať žiadnu aktivitu. Na Obrázku 2.2 sú znázornené jednotlivé možné stavy, tvoriace životný cyklus aplikácie [1].

Z obrázka vyplýva, že na začiatku aplikácia nepracuje. Pri spustení je hlavné okno neaktívne a aplikácia má čas načítať potrebné dáta. Vzhľadom na nepresný názov stavu je nutné poznamenať, že neaktívna aplikácia v skutočnosti stále pracuje na popredí, ale nedostáva žiadne udalosti. Po načítaní dát je aplikácia prezentovaná používateľovi a sú jej posielané udalosti. Z tohoto stavu sa ale kedykoľvek môže dostať do pozadia, avšak musí znova prejsť cez neaktívny stav. Na pozadí sa potom od aplikácie očakáva, že nebude pracovať alebo pri určitých výnimkách bude pracovať len minimálne. Ďalším možným stavom je pozastavenie, do ktorého sa aplikácia dostáva, pokiaľ nie je možné vykonávať jej kód.



Obr. 2.2: Životný cyklus aplikácie prevzatý z dokumentácie [1].

Priamo podporované architektúry aplikácií

Spoločnosť Apple Inc. priamo podporuje v rámci Swiftu dve architektúry *Model-View-Controller* (MVC) a *Model-View-ViewModel* (MVVM). Obecne ide o rozdelenie systému na tri časti. Prvou je rámec, ktorý sa stará o uchovávanie a transformáciu dát. Druhou časťou je niečo, čo dáta zobrazí. Navyše sa druhá časť stará o jednotlivé vstupy používateľov do systému. Posledná časť býva akýmsi mostom medzi prvými dvoma časťami.

MVC je jedným z najpoužívanejších typov architektúr systémov, a tak bol vo Swifte podporovaný už od začiatku. Túto podporu má pre iOS na starosti knižnica UIKit. Programátori ale v tejto architektúre často vytvárali príliš obsiahly ovládač (Controller) a často narušovali správne rozdelenie aplikácie na moduly. V roku 2019 ale prišla knižnica *SwiftUI*, ktorá priniesla priamu podporu pre MVVM. Rozdielom je, že neexistuje *Controller*. Používateľské rozhranie je v tejto architektúre popísané ako stav aplikácie [13]. Zobrazované okná sú zložené z inštancií štruktúry, ktorá implementuje protokol `View`. Tieto štruktúry vytvárajú pohľady. Aby mohlo byť okno dynamické, a teda aby ho bolo možné prekresliť na základe nejakej udalosti, môžeme dať jednotlivým pohľadom špeciálne atribúty. Tieto atribúty využívajú dvojcestné previazanie (Binding) s *ViewModelom* alebo priamo s *Modelom*. Stav potom chápeme práve ako súhrn hodnôt viazaných atribútov. Tento stav je uložený vždy v objekte *ViewModel*, ktorý sa stará o zobrazovanie okien a ich pohľadov. Pre prekreslenie okna sa musí zmeniť stav aplikácie, čiže sa musí zmeniť aspoň jeden atribút naviazaný na nejaký pohľad. Je nutné podotknúť, že sa neprekresľuje vždy celé okno, ale iba jeho pohľady, na ktoré sú naviazané zmenené stavy.

Knižnica SwiftUI

Knižnica SwiftUI predstavuje protokol známy ako pohľad (`View`), ktorý musí byť adaptovaný štruktúrou aby mohla byť zobrazená. Protokol vyžaduje definíciu atribútu `body`, ktorého hodnota taktiež implementuje protokol View. Výpis 2.3 ukazuje vytvorenie jednoduchého pohľadu, ktorý obsahuje text „Hello, World!“.

```
1 // Struct must implement View protocol
2 struct HelloWorldView: View {
3   var body: some View { // Variable needed by protocol View
4         Text("Hello, Wordl!") // View printing string to display
5   }
6 }
```

Výpis 2.3: Definovanie jednoduchého pohľadu, obsahujúceho text *„Hello, World!“*

Spôsoby sledovania zmien atribútov pre prekreslenie pohľadu

Dynamika sa zabezpečí pridaním atribútov, ktoré musia byť obalené obaľovačom vlastností (property wrapper, v nasledujúcom texte sa používa iba výraz wrapper) špeciálneho typu. Zmena hodnôt wrapperov sa potom bude propagovať do `ViewModelu`, čím sa zaručí prekreslenie okna. Paul Hudson vo svojom článku [6] vysvetľuje jednotlivé spôsoby sledovania zmien atribútov.

Najdôležitejším wrapperom je `@State`, ktorý dokáže obaliť atribúty jednoduchých dátových typov a štruktúry. Referencované typy sa sledujú pomocou wrappera `@ObservedObject`, ktoré musia implementovať protokol `ObservableOject`. Pre označenie sledovaných atribútov v referencovanom type existuje wrapper `@Published`. Referencované typy je možné sledovať aj pomocou wrappera `@StateObject`, pri ktorom je za životný cyklus zodpovedná aplikácia a nie priamo používateľ, ako to je pri `@ObservedObject`. Ďalším wrapperom je `@EnviromentObject`, ktorý umožňuje získať objekty, implementujúce protokol `ObservableObject`, ktoré boli vytvorené pohľadmi v hierarchicky vyššej úrovni. Vďaka tomu nemusíme tieto objekty predávať argumentami cez niekoľko volaní pohľadov, čo zlepšuje celkovú čitateľnosť kódu.

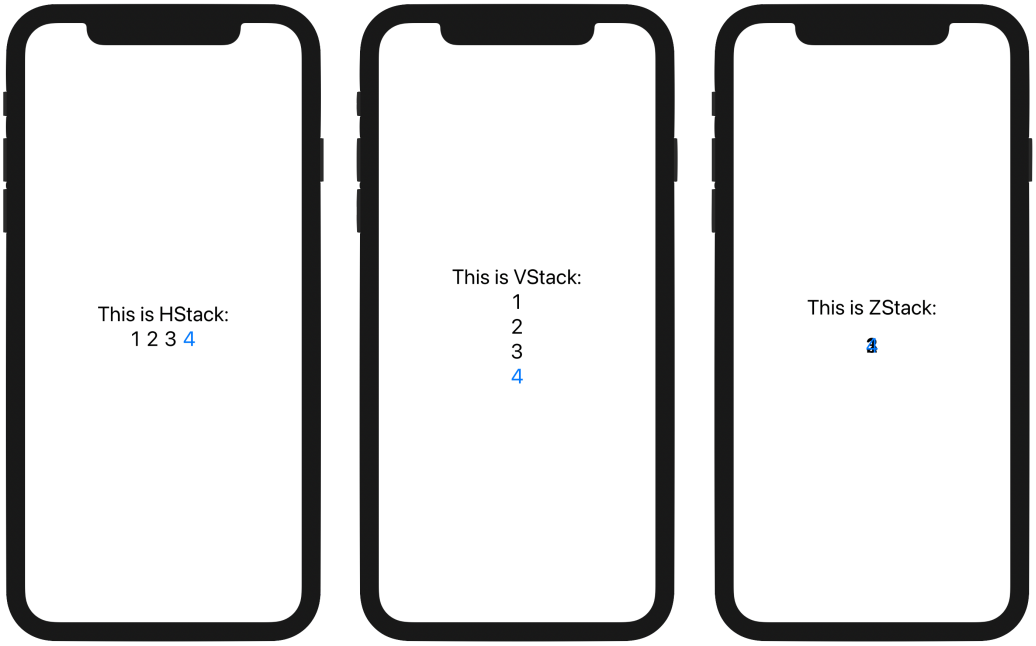
Rozloženie pohľadov

SwiftUI poskytuje tri základné konštrukcie pre rozloženie okien: `HStack`, `VStack` a `ZStack`. Slúžia k ukladaniu pohľadov vedľa seba na základe špecifikovanej osi. `HStack` pokladá pohľady vedľa seba (zľava doprava), `VStack` ich pokladá pod seba (zhora dole) a `ZStack` cez seba (od zadu do popredia). Obrázok 2.3 zobrazuje postupne jednotlivé pohľady vytvorené Výpisom 2.4, pričom vo výpise je nutné nahradiť `PLACEHOLDER` za jednotlivé názvy pohľadov.

```
1 struct DemoStackViews: View {
2   var body: some View {
3     VStack {
4       Text("This is PLACEHOLDER:")
5       PLACEHOLDER {
6         Text("1")
7         Text("2")
8         Text("3")
```

```
9           Text("4").foregroundColor(.blue)
10        }
11      }.font(.title)
12    }
13  }
```

Výpis 2.4: Šablóna pre zobrazenie rozdielu medzi `HStack`, `VStack` a `ZStack`.



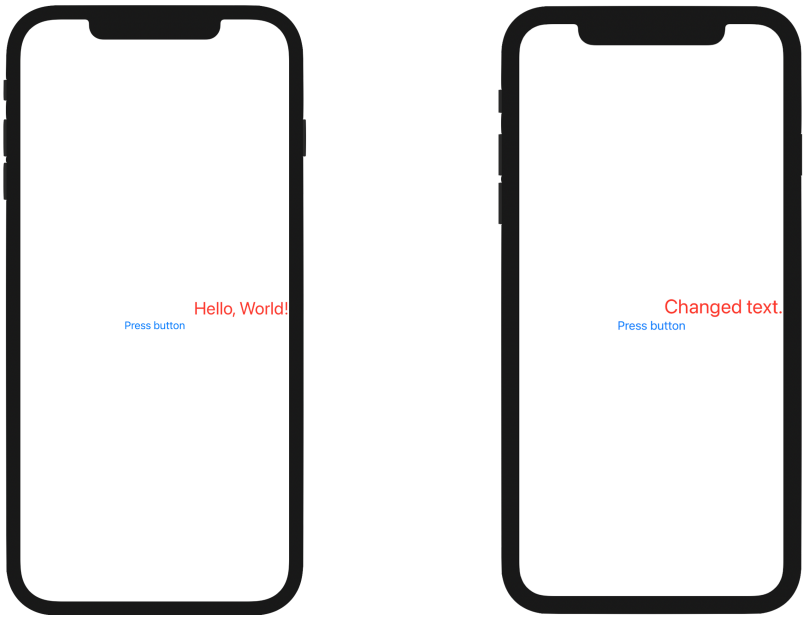
Obr. 2.3: Demonštrácia rozloženia pohľadov. Zľava doprava sú použité rozloženia `HStack`, `VStack` a `ZStack`.

Základné pohľady používané v knižnici SwiftUI

Namiesto štruktúry `Text` môžeme vložiť do tela premennej `body` iné zložitejšie štruktúry a tak vytvárať zložitejšie okná a pohľady. Pre umožnenie zadávania textu používateľom sa používa okno `TextField`. Toto okno pomocou parametra, ktorého typ je `Binding<String>`, dokáže uložiť vstup od používateľa do premennej poskytnutej parametrom. Pre jednoduchý predom definovaný používateľský vstup je pripravené tlačítko, reprezentované štruktúrou `Button`. Táto štruktúra vyžaduje dva uzávery. Prvý definuje, čo sa má stať po stlačení tlačidla – akciu a druhý definuje obsah tlačidla.

Štylizácia pohľadov

Štruktúry je možné formátovať a meniť ich výsledný vzhľad. Napríklad štruktúra `Text` môže byť formátovaná metódami meniacimi vzhľad textu. Takouto metódou je napríklad metóda `font` meniaca veľkosť písma alebo `foregroundColor` meniaca farbu písma. Pre hrubé písmo sa zase používa `fontWeight`. Farba pozadia sa mení pomocou metódy `background`. Zmena veľkosti pohľadu je možná metódou `frame`. Jednoduchú štylizáciu zobrazenú na Obrázku 2.4 je možné vytvoriť napríklad pomocou Výpisu 2.5.



Obr. 2.4: Demonštrácia okna pred a po stlačení tlačítka.

```
struct DemoPushButton: View {
  // If variable is changed redraw view
  @State var stringVariable = "Hello, World!"
  var body: some View {
    VStack {
      HStack{
        Spacer() // Puts all other views to right
        // Draws content of stringVariable on display
        Text(stringVariable)
          .font(.title) // Changes font of text to title
          .foregroundColor(.red) // Changes text color to red
      }
      // Draws button struct into display
      Button(action: { // Closure executed after button pressing
        stringVariable = "Changed text."
      }, label: { // Represents what button looks like
        Text("Press button")
      })
    }
  }
}
```

Výpis 2.5: Štruktúra popisujúca zložitejšie okno s formátovaným textom a tlačítkom.

Navigácia medzi pohľadmi

SwiftUI podporuje dve základné navigačné prostriedky. Prvým je `NavigationView`, ktorý spolupracuje s `NavigationLink`. Spolu umožňujú vytvorenie pseudo zásobníka pohľadov.

Pričom je zobrazený iba ten pohľad, ktorý je na vrchole zásobníka. Na tomto zásobníku ako prvým a neodstrániteľným prvkom je pohľad definovaný telom `NavigationView`. `NavigationLink` je potom akýmsi prostredníkom (tlačidlom), ktorým používateľ dokáže na vrch zásobníka pridať okno definované prvým argumentom pri inicializácii pohľadu `NavigationLink`. Táto dvojica avšak obecne pridáva aj prostriedok na odstránenie okna z vrchu zásobníka.Týmto prostriedkom je tlačidlo zobrazované v ľavom hornom rohu.

Druhá varianta navigácie je `TabView`, ktorá vytvára spodnú lištu s odkazmi na pohľady definované v jej tele. Odkazom sa potom zobrazí priradený pohľad spolu s touto spodnou lištou.

2.4.3 Tvorba serverových aplikácií pomocou knižnice Vapor

Knižnica Vapor bola vytvorená pre tvorbu aplikácií na serverovej strane. Ide o knižnicu s expresívnym, protokolárne orientovaným dizajnom zameraným na typovú bezpečnosť. Vapor priamo podporuje autentifikáciu pomocou algoritmov *Basic* alebo *Bearer*. Ďalej podporuje prácu s databázovými systémami ako *MySQL*, *PostgreSQL*, *SQLite* a *MongoDB*. Dovoľuje vytvárať HTML šablóny, ďalej umožňuje vytvárať klientské a serverové aplikácie využívajúce HTTP a mnoho ďalších konceptov.

Základné princípy

Tak ako každá serverová aplikácia, aj aplikácia vytvorená v knižnici Vapor čaká v nekonečnej slučke na dotazy. Vapor k tomu využíva protokol `EventLoop`, ktorý reprezentuje nekonečný cyklus spracovávajúci vstupné a výstupné procesy. Pre zaistenie spracovania čo najviac požiadaviek prichádzajúcich v podobnom čase je možné obsluhovať procesy z viacerých vlákien. Práca s procesmi je postavená na modernom princípe budúcnosť/prísľub (Future/Promise), ktorý zjednodušuje prácu s viacerými vláknami a asynchrónnymi procesmi.

Tento princíp funguje tak, že asynchrónny proces sa vykonáva na inom vlákne, pričom v aktuálnom vlákne sa vytvorí objekt (`EventLoopFuture`) reprezentujúci výsledok asynchrónneho kódu získaného niekedy v budúcnosti. Nad týmto objektom je možné volať ďalšie funkcie, a tým určiť ďalšie spracovanie asynchrónneho výsledku. Medzi základné metódy patria metóda `map`, ktorá transformuje prisľúbenú hodnotu na iný typ a metóda `flatMap`, ktorá transformuje prisľúbenú hodnotu na inú inštanciu typu `EventLoopFuture`. Tento prístup priamo nadväzuje na funkcionálne programovanie, a teda pristupovanie k všetkým prvkom programu ako k funkciám.

Smerovanie požiadaviek

Smerovanie (Routing) je v tomto kontexte vytváranie ciest. Cestám sa určujú identifikátory (časti cesty oddelené symbolom `/`) a im priradená funkcionalita. V knižnici Vapor sa toto robí pomocou registrácie jednotlivých ciest. Registrácia pozostáva z určenia HTTP metódy, určenia identifikátora a definície samotnej funkcie, tak ako je zobrazené na Výpise 2.6.

Výpis 2.6 obsahuje prvok `app`, ktorý je inštanciou triedy `Application`, a teda reprezentuje serverovú aplikáciu. Za použitia jej metódy `get` sa určuje blok kódu pre obsluhu HTTP správ typu *GET* s cestou *HelloWorld*. Metóda umožňuje vytvárať zložitejšie cesty pomocou variabilného počtu argumentov. Posledným argumentom je uzáver, ktorého parameter je objekt typu `Request`. Tento objekt reprezentuje prijatú HTTP požiadavku. Uzáver musí vrátiť hodnotu typu, ktorý implementuje protokol `Content`, a teda je možné vrátený typ

reprezentovať ako HTTP odpoveď. Vrátená hodnota je potom serializovaná a odoslaná ako odpoveď na HTTP požiadavku. Popísaný výpis vracia HTTP odpoveď s kódom *200 OK* a telom obsahujúcim reťazec *Hello, World!*.

```
app.get("HelloWorld") { req in
  // Returned value will be encapsuled and send as body of HTTP answer
  return "Hello, World!"
}
```

Výpis 2.6: Registrácia identifikátora *HelloWorld*.

Vapor umožňuje aj generické identifikátory, ktorých názvy musia byť pomenované a majú prefix `:` (dvojbodkou). Príkladom pre generický identifikátor môže byť identifikátor pomenovaný `:id`, vďaka čomu je možné z objektu `Request` získať hodnotu identifikátora pod kľúčom `id"`. Protokol `Content` je spojením protokolov `Codable`, `RequestDecodable` a `ResponseEncodable`. Používajú sa pri kódovaní a dekódovaní dát posielaných po sieti.

Vytvorenie databázy

Pre prístup k databáze je nutné nastaviť konfiguráciu, to spočíva v zadaní prihlasovacích údajov, názov databázy a jej umiestnenie. Vapor sa pomocou tejto konfigurácie potom postará o pripojenie na databázu. Vapor umožňuje definovať a vytvoriť prvky databázy ako sú tabuľky alebo počiatočné dáta pomocou *migrácií*. Úlohou migrácie je vytvoriť alebo upraviť tabuľku tak, aby vhodne reprezentovala požadované objekty. Vo Výpise 2.7 je zobrazená migrácia pre vytvorenie tabuľky `demoTable`. Vo výpise sa postupne nastaví meno tabuľky, potom sa vytvoria dva stĺpce a nakoniec sa vytvorí celá tabuľka.

```
struct CreateDemoTable: Migration {
  // Function is called on migration
  func prepare(on database: Database) -> EventLoopFuture<Void> {
          // Returns promise of creation
      return database.schema("demoTable")
          .id() // Adds identifying column
          // Adds required column named "name" of string type
          .field("name", .string, .required)
          .create() // Creates table
  }
}
```

Výpis 2.7: Jednoduchá štruktúra, ktorá pri migrácii vytvorí tabuľku `demoTable`.

Reprezentácia záznamov v tabuľke

K prístupu k jednotlivým riadkom tabuľky sa používa *model*, ktorý reprezentuje jeden záznam a jeho vzťahy. Vo Výpise 2.8 je uvedený jednoduchý model pre tabuľku *demoTable*, ktorá má dva stĺpce: identifikátor *id* a meno *name*. `DemoModel` je konečnou triedou (final class), ktorá adaptuje protokoly `Model` a `Content`. Trieda definovaním premennej `schema` určuje, ku ktorej tabuľke budú inštancie triedy patriť. Premenná `id` je obalená wrapperom `@ID`, čím sa označuje privátny kľúč. Druhá premenná je obalená wrapperom `Field`, ktorý má za úlohu reprezentovať obyčajný stĺpec tabuľky.

```swift
final class DemoModel: Model, Content {
  static let schema = "demoTable" // Sets name of source table
  // Sets identifier to be referenced from id variable
  @ID(key: .id) var id: UUID?
  @Field(key: "name") var name: String // Represents column "name"
  init() {} // Needed by protocol
  // Creates DemoModel object
  init(id: UUID? = nil, name: String) {
    self.id = id
    self.name = name
  }
}
```

Výpis 2.8: Deklarácia jednoduchého modelu.

Relácie medzi záznamami

Relácie sú riešené wrappermi `@Parent`, `@Child`, `@OptionalChild`, `@Children` a `@Siblings`. Tabuľka 2.1 zobrazuje možné relácie a nutné wrappere pre ich vytvorenie v knižnici Vapor. Pri jednotlivých wrapperoch je naznačený aj typ obalených atribútov. Ľavá strana relácie ukazuje počet odkazovaní *Modelu A* na *Model B*. Pravá zase opačnú stránku relácie. Stĺpce modelov zobrazujú obaľovače a typy obalených premenných, použité daným modelom pre správne vytvorenie vzťahu. *Model C* vyjadruje pomocný model reprezentujúci pomocnú tabuľku.

Model A	Relation	Model B	Model C
@Parent B	1:0..1	@OptionalChild A	Not needed
@Parent B	1:1	@Child A	Not needed
@Parent B	1:M	@Children [A]	Not needed
@Siblings [C]	M:N	@Siblings [C]	@Parent A, @Parent B

Tabuľka 2.1: Využitie wrapperov vzhľadom k reláciám.

Práca s databázou

Vapor ponúka metódy pre prácu s databázou priamo v protokole `Model`. Metódy slúžia k vytvoreniu databázovej požiadavky a vracajú typ `QueryBuilder`. Záznamy je možné filtrovať pomocou funkcie `filter`, ktorá ako parameter berie predikát, ktorý musia vrátené záznamy spĺňať. Nakoľko spracovanie databázovej požiadavky je asynchrónne spracovávané, tak výsledok spracovávania je typu `EventLoopFuture`. Prísľub na výsledné záznamy z databázy je možné získať napríklad metódami `first` (prvý záznam) alebo `all` (všetky záznamy).

Kapitola 3

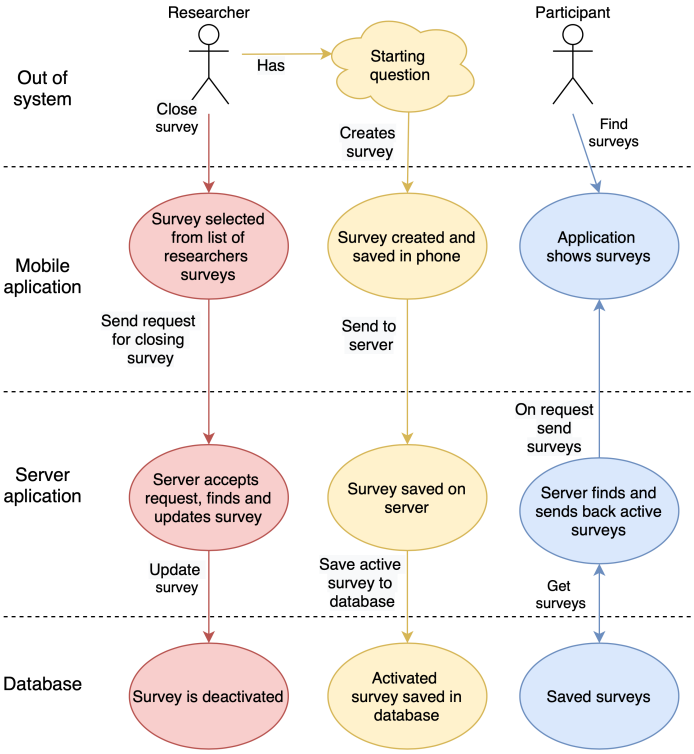
Návrh jednotlivých častí systému

Zo skúseností získaných z praktickej činnosti programátora vyplýva, že najdôležitejšou časťou pri tvorbe systému je jeho samotný návrh. Zlý návrh vedie ku zlému výsledku, ktorý je potom nutné prerábať. Súčasťou môjho systému budú dve aplikácie. Prvou bude aplikácia pre mobilné zariadenia a druhou bude serverová aplikácia. Návrh mobilnej aplikácie spočíva v určení funkcionality systému vzhľadom na používateľa. Jednotlivé prípady používania potom nepriamo určujú časti grafického užívateľského rozhrania. Veľkou súčasťou návrhu serverovej aplikácie je aj návrh databázy. Serverová aplikácia prepája komunikáciu medzi databázou, mobilnou aplikáciou a aplikáciou predajcu. Predajcom sa myslí používateľ, ktorý pridal do aplikácie aspoň jeden kupón.

3.1 Návrh životného cyklu dotazníku

Hlavným zmyslom systému je práca s dotazníkmi, preto je práca s nimi oporným bodom návrhu celého systému a jeho častí. Pre lepšie pochopenie potrieb dotazníkov a určenie špecifikácií častí systému bol navrhnutý životný cyklus dotazníku. Tento cyklus je znázornený na Obrázku 3.1. Obrázok je rozdelený na štyri časti, ktoré znázorňujú jednotlivé oblasti zodpovednosti. Vrchná časť zobrazuje zdroje a podnety patriace k vonkajšiemu svetu mimo systém. Druhá časť reprezentuje mobilnú aplikáciu nachádzajúcu sa na zariadení užívateľa. Predposledná časť je vyobrazená oblasť, ktorú má na starosti serverová aplikácia, a teda reprezentuje jej vnútornú pamäť a prácu. V spodnej časti je samostatná databáza.

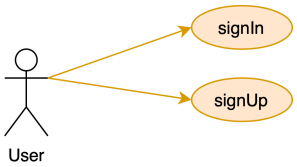
Každý životný cyklus dotazníku začína pri používateľovi. Najprv musí existovať túžba po získaní odpovede na nejakú počiatočnú otázku. S touto túžbou si používateľ otvorí aplikáciu, v ktorej začne vytvárať dotazník. Počas toho ako používateľ vytvára dotazník, budú všetky potrebné informácie ukladané do pamäti aplikácie. Po vložení otázok používateľ potvrdí, že chce vytvoriť dotazník. Potvrdený dotazník bude odoslaný na server. Prijatý dotazník serverová aplikácia uloží do databázy, a tak sa stane dostupným pre všetkých relevantných používateľov. Uložený dotazník sa potencionálnym respondentom zobrazuje na hlavnej stránke. Požívatelia môžu vyplňovať dotazník, pokiaľ ho tvorca dotazníku neuzatvorí. Bádateľ si bude môcť priebežne prehliadať výsledky dotazníkov a keď uzná za vhodné, pomocou aplikácie uzatvorí dotazník. Častým prípadom používania systému môžu byť prieskumy, ktoré ale potrebujú nejako podložiť svoje závery. Práve preto bude mať bádateľ možnosť si nechať exportovať výsledky dotazníku.



Obr. 3.1: Životný cyklus dotazníka.

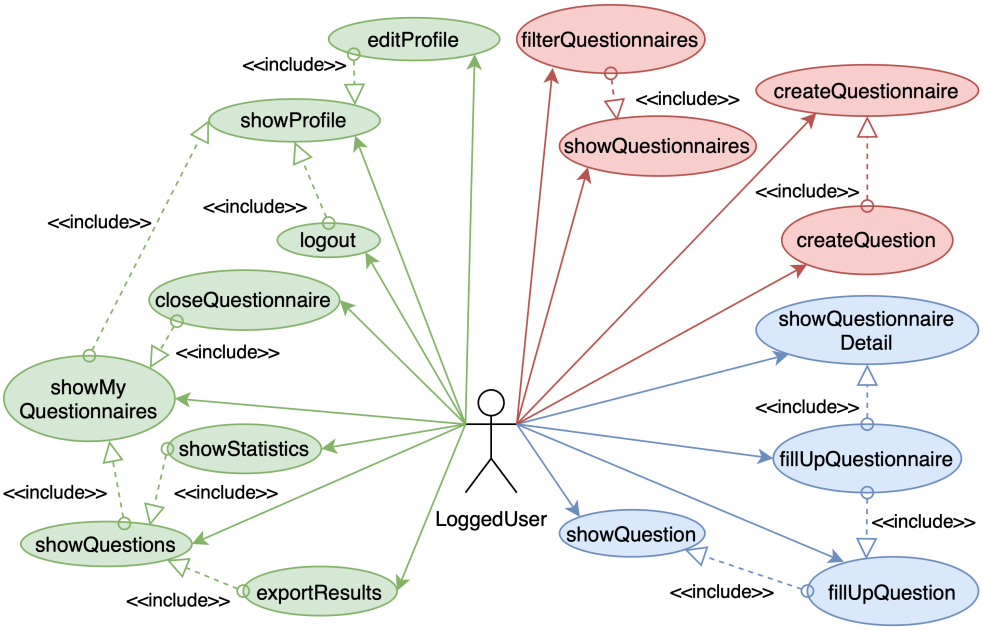
3.2 Návrh mobilnej aplikácie

Hlavnou úlohou mobilnej aplikácie bude zobrazovať dáta získané zo servera alebo v prípade vyplnenia dotazníka, odoslanie jeho odpovedí na server. Túto funkcionalitu systému popisujem pomocou diagramov prípadov použitia (ďalej len UCD). Pre zaistenie toho, že jeden používateľ nevyplnil rovnaký dotazník viac krát, sa používatelia musia do systému prihlásiť. Táto skutočnosť rozdeľuje používateľov na registrovaných a neregistrovaných, pričom vďaka tomuto je možné obmedziť funkcionalitu aplikácie, ktorú používa neregistrovaný používateľ. Po spustení aplikácie sa preto používateľ môže do aplikácie iba prihlásiť alebo poprípade registrovať. Toto ukazuje Obrázok 3.2.



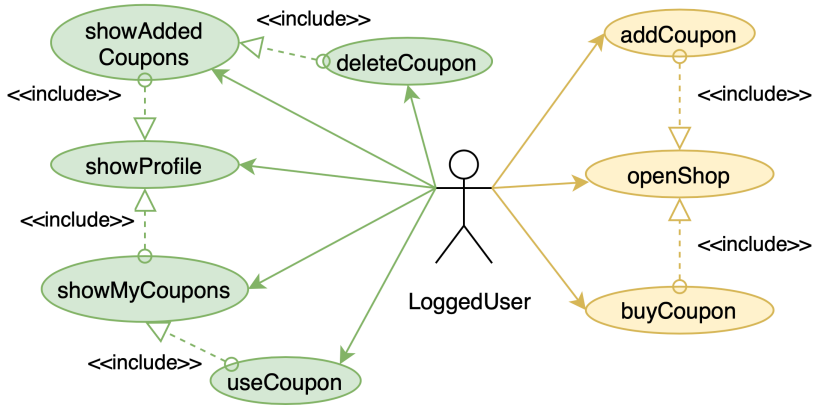
Obr. 3.2: UCD Neprihláseného užívateľa.

Prihlásení používatelia budú môcť prehľadávať dostupné dotazníky a vyplňovať ich. Budú môcť taktiež bez ďalších formalít vytvárať nové dotazníky. Vytváranie dotazníku bude zahrňovať postupné pridávanie otázok. Bádatelia majú možnosť prezerania výsledkov dotazníkov, ktoré vytvorili. Tieto a ďalšie možnosti používania aplikácie a ich prepojenia sú naznačené v prvej časti Obrázku 3.3.



Obr. 3.3: UCD Prihláseného užívateľa časť 1/2

Ako motiváciu bude každý používateľ dostávať za vyplnenie dotazníkov tokeny. Tie si bude môcť v obchode vymeniť za kupóny. Po zakúpení kupónu bude kupón zobrazený v profile používateľa, z kade ho môže používateľ aktivovať. Každý používateľ taktiež môže pridať do systému nový kupón, je ale nutné pri tom zadať ďalšie potrebné informácie. Obrázok 3.4 je druhou časťou diagramu prípadov použitia prihláseného používateľa a zobrazuje akty súvisiace s prácou s tokenmi.

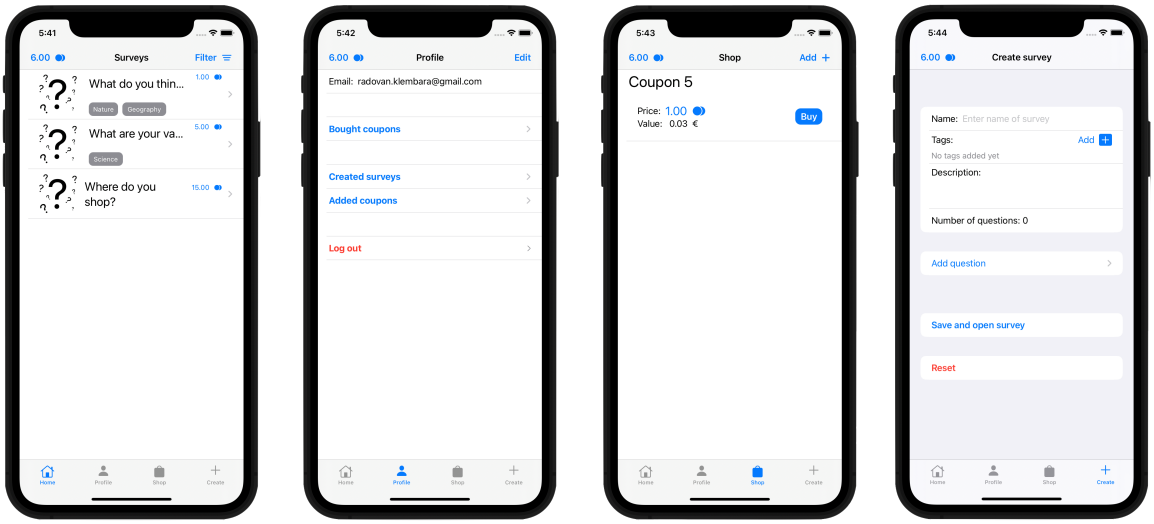


Obr. 3.4: UCD Prihláseného užívateľa časť 2/2

Grafické používateľské rozhranie aplikácie

Pri prvom otvorení aplikácie sa používateľovi zobrazí prihlasovacie okno s možnosťou prihlásiť sa alebo registrovať. Registrácia alebo neúspešné prihlásenie bude oznámené pomocou modálnych okien.

Po prihlásení do aplikácie je používateľovi zobrazená hlavná stránka. Na vrchu obrazovky sa zobrazuje horná lišta, ktorá slúži používateľovi pre lepšiu navigáciu aplikáciou. V ľavom rohu lišty sa zobrazuje aktuálny počet tokenov okrem prípadov, kedy navigácia vyžaduje zaplniť celý priestor. Pravý horný roh je využívaný pre rôzne tlačidlá a ich funkcie. V spodnej časti obrazovky je menu odkazujúce sa na štyri základné okná aplikácie. Prvým oknom je hlavná stránka. Ďalšími oknami sú okná profilu, obchodu a okno pre vytvorenie dotazníku. Návrh hlavných okien je zobrazený na Obrázku 3.5



Obr. 3.5: Návrh hlavných okien aplikácie.

Na hlavnej stránke sa zobrazujú používateľovi dostupné dotazníky. Zobrazované sú iba počiatočné informácie ako je názov dotazníku, odmena za jeho vyplnenie a poprípade aj označenia tematických okruhov dotazníku. Po vybraní dotazníku sa zobrazia podrobnejšie informácie, ako napríklad popis dotazníku a počet otázok. V tomto okne sa taktiež nachádza tlačidlo pre začatie vypĺňovania dotazníku. Okno otvorené týmto tlačidlom zobrazuje ukazovateľ postupu, znenie aktuálnej otázky a tlačidlá pre pokračovanie vo vypĺňaní.

Obrazovka profilu zobrazuje emailovú adresu používateľa. Ďalej sa z tejto obrazovky dá dostať k zoznamu vytvorených dotazníkov. Po vybratí konkrétneho dotazníku, bude zobrazený zoznam všetkých otázok dotazníka. V tomto okne sa v pravom hornom rohu nachádza tlačidlo, ktorým je možné exportovať výsledky dotazníku. Vybratím otázky zo zoznamu sa presunie používateľ do okna zobrazujúceho výsledky otázky. Výsledky sú zobrazované ako zoznam jednotlivých možností a ich percentuálneho zastúpenia alebo ako zoznam odpovedí. Pokiaľ sa jedná o otázku s výberom z možností, sú výsledky reprezentované aj v podobe grafu. V profile sa nachádza tlačidlo pre zobrazenie zakúpených kupónov. V tomto okne je možné aktivovať kupóny. Po úspešnom aktivovaní sa zobrazí zľavový kód. Okno profilu slúži aj na odhlásenie používateľa.

Obchod aplikácie je treťou hlavnou stránkou a zobrazuje všetky dostupné kupóny. Pri každom kupóne je zobrazený jeho názov, cena v tokenoch a hodnota v eurách. Kupóny je možné kupovať rovno z tohoto okna. Zakúpenie kupónu je podmienené potvrdením akcie v modálnom okne a dostatočným množstvom tokenov. Do obchodu môžu prispieť všetci používatelia, a to pridaním kupónov pomocou tlačidla. Pridanie zahŕňa vyplnenie potrebných informácií a samotné uloženie.

Vytvorenie dotazníku má na starosti posledná hlavná stránka. Stránka zobrazuje textové polia pre vyplnenie základných informácií ohľadom dotazníka ako názov alebo popis. Pre pridanie otázky sa využíva tlačidlo otvárajúce nové okno, v ktorom sa nachádza textové pole pre znenie samotnej otázky a nástroj na vybratie typu otázky, na základe ktorej sa bude meniť okno tak, aby používateľ mohol vyplniť dôležité časti pre vytvorenie konkrétneho typu otázky.

Pokiaľ pôjde o otázku s výberom z možností, tak sa zobrazí zoznam možných odpovedí, ktoré bude môcť používateľ upraviť. Otázka s otvorenou odpoveďou má mať iba jedno pole do ktorého používateľ môže vložiť prípadnú možnú odpoveď pre spresnenie možnej odpovedi respondentovi. Čo sa týka kombinovaných otázok, tak sa jedná o zoznam možností rozšírených o možnosť odpovedať otvorene, pričom ale bádateľ musí vyplniť názov tejto otvorenej možnosti. Pokiaľ pri ukladaní otázky dôjde k zisteniu chybnej informácie, je táto skutočnosť oznámená modálnym oknom bádateľovi. Po pridaní otázky sa zobrazia základné informácie v zozname otázok. Jednotlivé otázky je možné ako aj editovať, tak aj mazať.

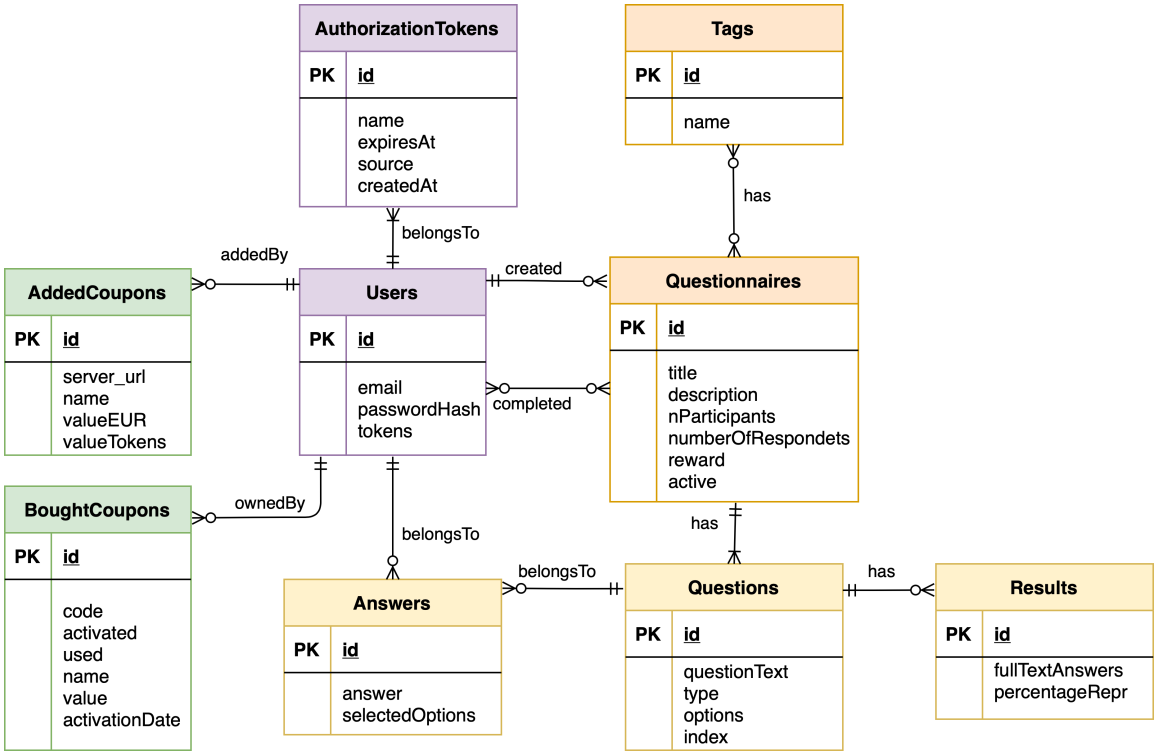
Pre samotné vytvorenie alebo prípadné zmazanie dotazníku bude pripravené tlačidlo. Vymazanie dotazníka bude podmienené potvrdením akcie v vyskakovacom okne. Pokiaľ dôjde k nejakej chybe, je táto skutočnosť reprezentovaná vyskakovacím oknom.

3.3 Návrh serverovej časti systému

Serverová časť systému je zložená z dvoch častí, a to zo serverovej aplikácie a databázy. Serverová aplikácia má na starosti komunikáciu medzi databázou a mobilnou aplikáciou. Pre komunikáciu s okolitým svetom bude server používať protokol HTTP. Ďalšou úlohou aplikácie je pripraviť databázu pri prvom spustení napríklad tak, že vytvorí tabuľky a relácie v databáze. Potom naplní tabuľku označení tematických okruhov dotazníkov počiatočnými dátami. Ďalej má na starosti správu výsledkov dotazníkov. Zabezpečuje získavanie zľavových kódov pri zaobstarávaní kupónov, aktiváciu a deaktiváciu jednotlivých kupónov. Ako databázový systém je zvolený systém PostgreSQL. Väčšina zo serverovej časti systému bude zastrešená webovým rámcom Vapor.

Návrh databázy

Databáza slúži najme na ukladanie práce s dotazníkmi. Medzi dotazníkmi a používateľmi existujú dve väzby. Prvá hovorí o tom, že používateľ vytvoril daný dotazník a druhá zaznamenáva, ktorý dotazník vyplnil ktorý používateľ. Nakoľko druhá väzba bude väzbou mnoho ku mnohým (používa sa skôr anglický preklad Many to Many alebo matematický zápis $M : N$) bude pre ňu vytvorená pomocná tabuľka. Každý dotazník je spojený taktiež s tabuľkou otázok. Toto spojenie hovorí o tom ktoré otázky patria ktorému dotazníku. Aplikácia podporuje päť typov otázok, čo je možné riešiť pomocou piatich veľmi podobných tabuliek. Nakoľko rozdielne hodnoty otázok sa nepoužívajú pri filtrovaní v databázových dotazoch, tak sú zakódované vo formáte JSON, a tak uložené v jednom stĺpci. Každá otázka je potom označená typom reprezentujúcim konkrétny typ otázky, aby bolo jasné, aká štruktúra je uložená. Ďalšou tabuľkou v databáze je tabuľka odpovedí. V tejto tabuľke sú uložené informácie o tom, aké odpovede zvolili jednotliví respondenti. Každému dotazníku môžu byť priradené označenia tematických skupín (tags). Priradenie skupiny dotazníku je taktiež väzbou *M:N*, preto je aj táto väzba v databáze reprezentovaná pomocnou tabuľkou. Overenie používateľov si vyžaduje tabuľku v databáze. Jej hlavnou úlohou je uloženie autorizačných tokenov, ktoré budú posielané používateľovi pri prihlásení do systému. Tieto tokeny budú



Obr. 3.6: ERD popisujúci databázu.

potom používatelia posielať v autorizačných hlavičkách HTTP dotazov smerujúcich na server. Súčasťou systému je aj obchod s kupónmi, ktoré sú taktiež uložené v databáze, nakoľko ich môže chcieť používateľ využiť aj neskôr ako ihneď po zakúpení. Obrázok 3.6, naznačuje celú databázu.

3.4 Návrh motivačného systému

Aby používatelia mali dôvod k používaniu systému, tak obsahuje aj motivačný systém. Stavebným kameňom tohoto systému sú *tokeny*, ktoré reprezentujú vnútroaplikačnú pseudo menu. Mena je navrhnutá tak, že minimálne ohodnotenie hodinovej práce človeka sú 3 eurá. To znamená že jedna minúta je ohodnotená 0.05 eurami. Najjednoduchším typom otázky na zodpovedanie je uzatvorená otázka s výberom z možností, ktorej vyplnenie trvá približne pol minúty. Pričom táto otázka je ohodnotená 1 tokenom, z toho vyplýva že 1 euro má hodnotu 40 tokenov.

Motivácia používateľov potom bude spočívať v získavaní týchto tokenov. Tokeny dostane každý používateľ automaticky za úspešné vyplnenie dotazníku. Každý dotazník preto bude po jeho vytvorení ohodnotený nejakým počtom tokenov. Pretože každý typ otázky si vyžaduje iný čas na jej vyplnenie, je každý typ ohodnotený inak. Jednotlivé hodnotenia otázok sú zobrazené v tabuľke 3.1.

Typy otázok	Odmena [token]
Zatvorená s výberom jednej možnosti	1
Zatvorená s výberom viacerých možností	2
Kombinovaná s výberom jednej možnosti	4
Kombinovaná s výberom viacerých možností	5
Otvorená	10

Tabuľka 3.1: Tabuľka hodnotení typov otázok.

Odmena za vyplnenie dotazníku je závislá na počte otázok a ich typoch. Dotazník je ohodnotený tak, aby platila rovnica (3.1) kde:

- T je celková odmena za vyplnenie dotazníku
- O je odmena za vyplnenie otvorenej otázky
- S_o je odmena za vyplnenie zatvorenej otázky s výberom jednej možnosti
- S_m je odmena za vyplnenie zatvorenej otázky s výberom viacerých možností
- C_o je odmena za vyplnenie kombinovanej otázky s výberom jednej možnosti
- C_m je odmena za vyplnenie kombinovanej otázky s výberom viacerých možností
- i je počet otvorených otázok
- j je počet uzatvorených otázok s výberom jednej možnosti
- k je počet uzatvorených otázok s výberom viacerých možností
- l je počet kombinovaných otázok s výberom jednej možnosti
- m je počet kombinovaných otázok s výberom viacerých možností

$$T = i * O + j * S_o + k * S_m + l * C_o + m * C_m \tag{3.1}$$

V aplikácii je obchod, v ktorom je možné kupovať zľavové kupóny za získané tokeny. Pridanie kupónu bude spočívať v zadaní názvu kupónu, jeho hodnoty a URL adresy, na ktorú bude možné zaslať HTTP požiadavku pre získanie zľavového kódu daného kupónu. Zároveň bude možné zadať hodnotu autorizačnej hlavičky HTTP požiadavky, aby bolo možné posielať aj autorizované požiadavky. Dodávateľ zadá hodnotu kupónu v eurách. Táto hodnota potom bude prepočítaná pomocou vzorca (3.2) kde:

- T je výsledná hodnota kupónu v tokenoch
- v je hodnota kupónu v eurách

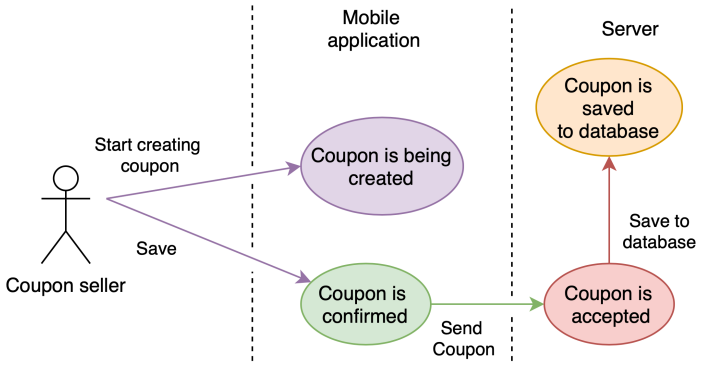
$$T = v * 40 \tag{3.2}$$

Po zakúpení je kupón dostupný pre aktiváciu. Pre použitie kupónu používateľ vyhľadá kupón medzi zakúpenými kupónmi a aktivuje ho. Aktivácia sprístupní zľavový kód na niekoľko minút. Uplatnenie zľavy bude mať na starosti predajca (dodávateľ zľavového kupónu).

Životný cyklus zľavového kupónu

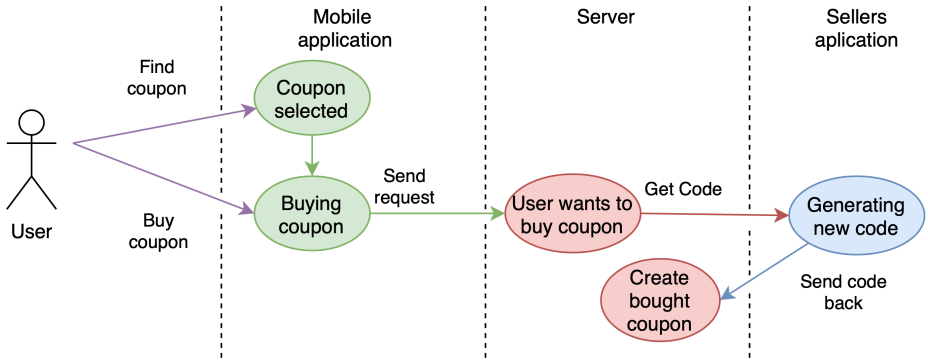
Životný cyklus bol navrhnutý tak, aby sa dal rozdeliť do troch častí. Každá z týchto častí reprezentuje nejakú skupinu úkonov týkajúcich sa práve práce s kupónmi. Pre ujasnenie, aké informácie sú potrebné k správnemu správaniu systému, bol pre zľavový kupón navrhnutý životný cyklus.

Obrázok 3.7 zobrazuje prvú časť cyklu. Ide o akt pridania kupónu. Systém očakáva, že používateľ, ktorý chce pridať kupón, má pripravené rozhranie pre získavanie zľavových kódov vo svojej webovej aplikácii. Toto rozhranie musí vedieť prijímať určenú HTTP požiadavku, na ktorú odpovie zľavovým kódom. V aplikácii si potom používateľ otvorí obchod a vloží potrebné informácie. Po vložení informácií používateľ potvrdí tvorbu kupónu a tým oznámi aplikácií, aby odoslala kupón na server. Prijatý kupón je spracovaný a až potom uložený do databázy.



Obr. 3.7: Životný cyklus zľavového kupónu časť 1/3, vytvorenie zľavového kupónu.

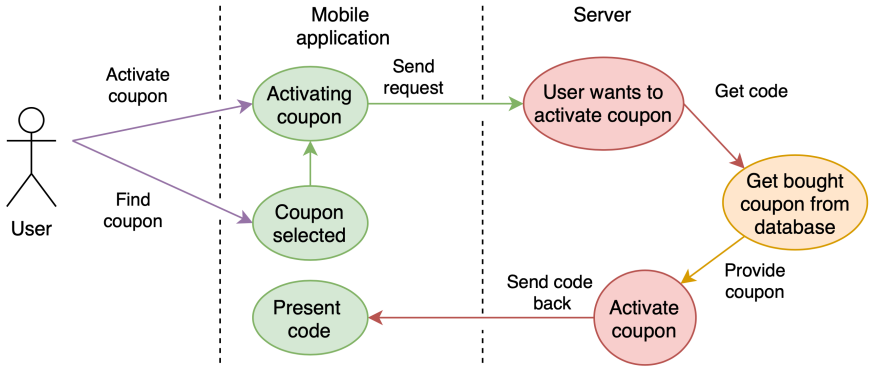
Druhá časť cyklu je naznačená Obrázkom 3.8. Reprezentuje kúpu kupónu. Pre kúpu kupónu používateľ potrebuje dostatok tokenov. Používateľ si zakúpi kupón, výberom kupónu zo zoznamu dostupných kupónov a potvrdením nákupu. Potvrdenie vyvolá odoslanie požiadavky na server, ktorý spracuje požiadavku tak, že sa pokúsi získať zľavový kód od predajcu pomocou HTTP požiadavky na predajcovu webovú adresu, ktorú zadal predajca pri tvorbe kupónu. Pri úspešnom získaní kódu je kúpený kupón uložený do databázy.



Obr. 3.8: Životný cyklus zľavového kupónu časť 2/3, kúpenie zľavového kupónu.

Obrázok 3.9 zobrazuje aktiváciu kupónu. Používateľ vyhľadá kupón medzi zakúpenými kupónmi a aktivuje ho. Aktivácia vyvolá zaslanie správy na server. Server označí kupón ako

aktivovaný, čím sa používateľovi sprístupní zľavový kód. Aplikácia automaticky zobrazí kód každého aktivovaného kupónu. Kód bude prístupný používateľovi 5 minút od jeho aktivácie.



Obr. 3.9: Životný cyklus zľavového kupónu časť 3/3, aktivácia zľavového kupónu.

Kapitola 4

Implementácia

Implementovaný systém je rozdelený podľa návrhu na dve časti, a to mobilnú a serverovú aplikáciu. Mobilná aplikácia je vytvorená pre zariadenia pracujúce s operačným systémom iOS, pričom sa vyžaduje verzia 14 alebo vyššia. Pre implementáciu bol použitý jazyk Swift verzie 5.3.2.

4.1 Mobilná aplikácia

Mobilná aplikácia bola vytvorená na základe návrhu primárne s využitím knižnice SwiftUI. Hlavná stránka zobrazená na Obrázku 3.5, ktorá zobrazuje dostupné dotazníky je vytvorená pomocou Výpisu 4.1[1]. Hlavná stránka sa skladá z navigácie (`NavigationView`) s textom „Surveys“, jednostĺpcovej tabuľky s viacerými riadkami (`List`) obsahujúcimi hlavný pohľad dotazníka (`MainPageSurveyPreview`) a tlačidlo pre filtrovanie dotazníkov podľa označení tematických celkov (`Button`).

```
1  struct BrowseNewSurveysView: View {
2    // ... Definitions of variables
3    @State private var questionnaires = [PublicQuestionnaire]()
4    @State private var isActive = [Bool]()
5    @State private var showSheet = false
6    var body: some View {
7      NavigationView {
8        Group {
9              // ... Check if there are any surveys
10             List {
11                 ForEach(questionnaires.indices, id: \.self) { i in
12                 // ... Check tags
13                 NavigationLink(destination:
14                   SurveyMainView(survey: questionnaires[i], rootIsActive:
                         self.$isActive[i]), isActive: self.$isActive[i]
15                 ){
16                   MainPageSurveyPreview(questionnaire: questionnaires[i])
17                 }
18               }
19             }
```

[1] Výpis je zjednodušenou verziou skutočnej implementácie. Komentáre indikujú vynechané časti.

```
20        }.navigationBarItems(leading: HStack {
21            Text(String(format: "%.2f", Defaults[.tokens]))
22            Image(systemName: "circlebadge.2.fill")
23          }.foregroundColor(.blue), trailing:
24            Button(action: { showSheet = true }){
25              HStack {
26                Text("Filter")
27                Image(systemName: "line.horizontal.3.decrease")
28              }.foregroundColor(.blue)
29        }).navigationBarTitle(Text("Surveys"), displayMode: .inline)
30      }
31      // ... Handle tag filtering & loading data
32    }
33  }
```

Výpis 4.1: Implementácia hlavnej stránky zobrazujúcej dostupné dotazníky.

Pohľad definujúci grafické rozhranie náhľadu jedného dotazníka je implementované Výpisom 4.2. Pre jednoduchosť sú vo výpise vynechané štylizačné prvky a kontrola existencie označení.

```
1  struct MainPageSurveyPreview: View {
2    @State var questionnaire: PublicQuestionnaire
3    var body: some View {
4      HStack {
5        Image("SurveyPlaceHolder")
6        VStack {
7          HStack {
8            Text(questionnaire.title)
9            VStack {
10             HStack{
11               Text(String(format: "%.2f", questionnaire.tokens))
12               Image(systemName: "circlebadge.2.fill")
13             }
14           }
15         }
16         HStack {
17           // ... Check if there are any tags
18           ScrollView(.horizontal, showsIndicators: false) {
19             HStack {
20               ForEach(questionnaire.tags, id: \.id) { t in Text(t.name) }
21             }
22           }
23         }
24       }
25     }
26   }
27 }
```

Výpis 4.2: Implementácia náhľadu dotazníka.

Komunikácia so serverovou aplikáciou

Pre komunikáciu so serverom bola vytvorená trieda `GeneralModel`, ktorá slúži ako rodičovská trieda pre triedy komunikujúce so serverovou aplikáciou. Pre samotnú komunikáciu bolo vytvorené rozšírenie triedy `URLSession`, ktoré sa stará o zaslanie požiadavky ako aj o získanie odpovedi. Obecne boli pre rôzne typy komunikácie vytvorené podtriedy triedy `GeneralModel` implementujúce protokol `ObservedObject`.

Poslanie požiadavky pomocou rozšírenia triedy URLSession

Výpis 4.3 zobrazuje zjednodušenú implementáciu rozšírenia `URLSession`. Na začiatku funkcie sa vytvorí objekt reprezentujúci požiadavku a nastaví sa HTTP hlavička. Pokiaľ parameter `body` nie je hodnota `nil`, tak sa po nastavení hlavičky enkóduje do formátu JSON a nastaví sa ako telo požiadavky. Rozšírenie využíva koncept reaktívneho programovania (knižnica *Combine*), vďaka čomu je spracovanie požiadavky jednoducho vyjadrené ako zreťazené posúvanie dát funkciám, ktoré ich spracovávajú. V tomto prípade sa najprv pošle požiadavka na server, po prijatí odpovedi sa následne skontroluje prípadný chybový kód HTTP požiadavky a nakoniec je telo odpovedi vrátené ako typ `Data`.

```
1  extension URLSession {
2    static func requestPublisher1<Upstream: Encodable>(method: HTTPMethod
         = .get, url: URL,
3      headers: [String: String], body: Upstream?) -> AnyPublisher<Data,
           Error> {
4
5      var request = URLRequest(url: url)
6      request.httpMethod = method.rawValue
7      if let body = body {
8        request.httpBody = try? JSONEncoder().encode(body)
9      }
10     return URLSession.shared.dataTaskPublisher(for: request)
11       .tryMap { output -> Data in
12         if let response = output.response as? HTTPURLResponse, response.
               statusCode != 200{
13           // ... Switch throwing some errors
14           throw HTTPError.statusCode
15         }
16         return output.data
17       }.receive(on: DispatchQueue.main).eraseToAnyPublisher()
18   }
19 }
```

Výpis 4.3: Rozšírenie triedy `URLSession`.

4.2 Serverová aplikácia

Serverová aplikácia bola písaná s využitím knižnice Vapor. Pre zaistenie správnej schémy databázy bolo nutné vytvoriť migračné štruktúry zodpovedné za správne vytvorenie tabuliek a väzieb medzi nimi. Ďalšou významnou implementovanou časťou je autorizácia prístupu k zdrojom aplikácie pomocou tried `User` a `Token`.

Vytvorenie databázy

Vytvorenie databázy sa vykonáva automaticky počas migrácie, ktorá je definovaná migračnými štruktúrami. Výpis 4.4 zobrazuje migráciu (vytvorenie) tabuľky pre otázky v dotazníku. Tabuľka identifikátor, tri stĺpce typu reťazec (`question_text`, `type`, `options`), jeden stĺpec (`is_part_of`) označujúci väzbu na tabuľku `questionaries` a stĺpec typu `Int` pre index. Všetky stĺpce tabuľky sú pomocou `.required` označené ako povinné. Vo výpise sa demonštruje vytvorenie rôznych stĺpcov, medzi ktorými je aj stĺpec naznačujúci väzbu medzi otázkou a dotazníkom.

```
struct CreateQuestion: Migration {
  func prepare(on database: Database) -> EventLoopFuture<Void> {
    return database.schema("questions")
      .id()
      .field("question_text", .string, .required)
      .field("type", .string, .required)
      .field("options", .string, .required)
      .field("is_part_of", .uuid, .references("questionnaires", "id"))
      .field("index", .int, .required)
      .create()
  }
}
```

Výpis 4.4: Štruktúra tvoriaca tabuľku otázok.

4.2.1 Autorizácia prístupu k zdrojom

Autorizácia je riešená pomocou autorizačných kódov (tokenov), ktoré sa zadávajú do hlavičky HTTP požiadavky. Autorizačný token je poslaný mobilnej aplikácii v odpovedi na prihlásenie. Aplikácia si tento token uloží a pridáva ho do hlavičiek ďalších požiadaviek. Každá požiadavka (okrem prihlasovacej a registračnej) musí obsahovať tento token.

Trieda Token

Táto trieda zaobaľuje tabuľku autorizačných tokenov. Validácia tokenov je potom implementovaná rozšírením triedy o protokol `ModelTokenAuthenticatable`. Implementácia rozšírenia je zobrazená vo Výpise 4.5.

```
extension Token: ModelTokenAuthenticatable {
  static let valueKey = \Token.$value
  static let userKey = \Token.$user
  var isValid: Bool {
    guard let expiryDate = expiresAt else {
      return true
    }
    return expiryDate > Date()
  }
}
```

Výpis 4.5: Rozšírenie triedy `Token` o protokol `ModelTokenAuthenticatable`.

Trieda User

Trieda `User` slúži k manipulácii so záznamami tabuľky používateľov. Trieda je rozšírená o funkciu `createToken`, ktorá vytvorí nový token s platnosťou tri mesiace a uloží ho do databázy. Funkcia je zobrazená vo Výpise 4.6.

```
1 extension User {
2   func createToken(source: SessionSource) throws -> Token {
3     let calendar = Calendar(identifier: .gregorian)
4     let expiryDate = calendar.date(byAdding: .month, value:3, to: Date())
5     return try Token(userId: requireID(), token: [UInt8].random(count: 16
          ).base64, source: source, expiresAt: expiryDate)
6   }
7 }
```

Výpis 4.6: Rozšírenie triedy `User` o funkciu pre vytvorenie nového autorizačného tokenu.

Povolenie prístupu len autorizovaným používateľom

Aby bolo možné autorizovať klienta na základe tokenu v HTTP hlavičke, bolo nutné označiť danú cestu a zároveň v rámci tela funkcie zavolať metódu pre autorizáciu tak, ako je znázornené na Výpise 4.7. Tento výpis je zjednodušenou verziou štruktúry `UserController`, ktorá registruje cestu pre získanie informácií o používateľovi.

```
1  struct UserController: RouteCollection {
2    func boot(routes: RoutesBuilder) throws {
3      let users = routes.grouped("users")
4      let tokenProtected = users.grouped(Token.authenticator())
5      let authUser = tokenProtected.grouped("authUser")
6      authUser.get(":id", use: getMyOwnUser)
7    }
8
9    func getMyOwnUser(req: Request) throws -> PublicUser {
10     try req.auth.require(Token.self).asPublic()
11     // Return user info
12   }
13 }
```

Výpis 4.7: Registrovanie identifikátora a jeho funkcionality, ktoré vyžadujú autorizovaný prístup.

4.2.2 Spracovanie požiadaviek

Serverová aplikácia pozná 21 rôznych požiadaviek. Požiadavky sú rozdelené medzi štruktúry kolekcií ciest pomocou štruktúr, ktoré implementujú protokol `RouteCollection`. Všetky cesty, okrem dvoch („signup“ a „login“), sú prístupné iba autorizovaným používateľom pomocou autorizačných tokenov.

Vytvorenie dotazníka

Výpisom 4.8 je implementované uloženie dotazníka do databázy. Dotazník aj s jeho otázkami je získaný z tela HTTP požiadavky. Následne sa vyhľadá bádateľ a vytvorí sa dotazník. K dotazníku sú potom pridané označenia tematických okruhov a samotné otázky.

```
1  func create(req: Request) throws->EventLoopFuture<PublicQuestionnaire>{
2    try req.auth.require(Token.self)
3    let cqStruct = try req.content.decode(CreateSurveyStruct.self)
4    // ... Get survey id (qID) and user id (id) from cqStruct
5    var questionnaire: EventLoopFuture<Questionnaire> = User.find(id, on:
         req.db).unwrap(or: Abort(.notFound)).flatMap { user in
6      let q = Questionnaire(id: qID, title: cqStruct.questionnaire.title,
           description: cqStruct.questionnaire.description, closeAfterDate:
           cqStruct.questionnaire.closeAfterDate, createdBy: user.id ?? UUID
           (), image: cqStruct.questionnaire.img, nQuestions: cqStruct.
           questionnaire.nQuestions, nRespondents: 0, token Double(cqStruct.
           questionnaire.tokens))
7      return user.$mySurveys.create(q, on: req.db).map { q }
8    }
9    // ...
10   var questionArr = [Question]()
11   for item in cqStruct.questions {
12     if let questionID = UUID(uuidString: item.id){
13       let question = Question(id: questionID, belongsToQuestionnaire: qID
             , qType: item.qType, qText: item.qText, qOptions: item.qOptions
             , index: item.index)
14       questionArr.append(question)
15     }
16   }
17   questionnaire = questionnaire.flatMap { q in
18     q.$questions.create(questionArr, on: req.db).map { q }
19   }
20   return questionnaire.flatMap { q in return q.asPublic(req: req) }
21 }
```

Výpis 4.8: Vytvorenie dotazníka a jeho následné uloženie do databázy.

Získanie kupónu

Zakúpenie kupónu zo strany serverovej aplikácie je implementované Výpisom 4.9. Najskôr sa z databázy získa profil bádateľa a potom je nájdený kupón, ktorý sa má kúpiť. Server získa zľavový kód zaslaním HTTP požiadavky na URL adresu dodanú predajcom. Následne je vytvorený nový záznam označujúci zakúpený kupón.

```
1  func buyCoupon(req: Request) throws -> EventLoopFuture<BoughtCoupon> {
2    try req.auth.require(Token.self)
3    let bStruct = try req.content.decode(BuyCouponStruct.self)
4    // ... Get coupon id (couponID) and user id (userID) from cqStruct
5    return User.find(userID, on: req.db).unwrap(or: Abort(.notFound)).
         flatMap { user in
```

```
6     return AddedCoupon.find(couponID, on: req.db).unwrap(or: Abort(.
          notFound)).flatMap { coupon in
7       // ... Check user tokens
8       let code = sendRequest(serverUrl: coupon.fromServer, token: coupon.
            token)
9       if code == "" {
10        return req.eventLoop.makeFailedFuture(Abort(.misdirectedRequest))
11      } else {
12        let b = BoughtCoupon(id: UUID(), code: code, ownedBy: userID,
              activated: false, used: false, name: coupon.name, value:
              coupon.value/Double(40))
13        user.tokens -= coupon.value
14        return user.update(on: req.db)
15          .flatMap { user.$ownedCoupons.create(b, on: req.db) }.map { b }
16      }
17    }
18  }
19  throw Abort(.badRequest)
20 }
```

Výpis 4.9: Zakúpenie zľavového kupónu.

Kapitola 5

Testovanie

Aby sa docielila správna funkcionalita systému, musel byť systém otestovaný. Pre toto testovanie bola zvolená technika simulovanej prevádzky. Išlo o sledovanie 14 používateľov pri práci s aplikáciou. Sledovaní používatelia mali za úlohu buď vytvoriť dotazník a po nejakom čase získať výsledky dotazníku alebo vyhľadať a vyplniť dotazník. Súčasťou testovania bolo aj vytvorenie kupónu a následné odskúšanie jeho funkcionality. Chyby odhalené pri testovaní boli následne opravené.

5.1 Príprava pre testovanie

Za účelom možnosti testovať systém bola serverová aplikácia a databáza distribuovaná pomocou služby *heroku*[1]. Mobilná aplikácia bola distribuovaná používateľom pomocou služby *TestFlight* od spoločnosti Apple.

Pre samotné testovanie bola vytvorená ďalšia jednoduchá serverová aplikácia, ktorej úlohou bolo simulovať aplikáciu predajcu. Pre aplikáciu bola vytvorená databáza, v ktorej sa uschovávajú kupóny a autorizačné tokeny (aby bolo možné testovať aj autorizovaný prístup ku kódu kupónu). Táto aplikácia spolu s databázou bola tak isto distribuovaná pomocou služby *heroku*.

Pre overenie spokojnosti s aplikáciou bol vytvorený dotazník A.

5.2 Testovanie pomocou simulovanej prevádzky systému

Pre simulovanie prevádzky boli testovací používatelia rozdelení do dvoch skupín podľa toho, čo bolo ich úlohou. Prvá skupina mala za úlohu vytvárať rôzne dotazníky, aby sa zistilo, či ich tvorba je jednoduchá a intuitívna. Ich druhou úlohou bolo pridávanie zľavových kupónov do aplikácie. Ďalšou úlohou bolo prezeranie výsledkov dotazníkov a ich prípadné exportovanie.

Druhá skupina skúšala vypĺňať dotazníky vytvorené prvou skupinou. Za vypĺňanie získavali tokeny, ktoré mali potom vymeniť za zľavové kupóny. Poslednou úlohou bolo získať zľavový kód.

[1] www.heroku.com

5.3 Závery vyvodené z výsledkov testovania

Po testovaní bol každému používateľovi zaslaný dotazník A. Tabuľka 5.1 a 5.2 zobrazuje výsledky odpovedí dotazníka. Z tabuliek vyplýva vysoká miera spokojnosti používateľov s aplikáciou.

Možnosti	Reprezentácia odpovedí [%]
Veľmi súhlasím	28,58
Súhlasím	42,85
Ani súhlasím, ani nesúhlasím	14,29
Nesúhlasím	7,14
Veľmi nesúhlasím	7,14

Tabuľka 5.1: Tabuľka hodnotení typov otázok.

Možnosti	Reprezentácia odpovedí [%]
Áno	78,57
Nie	21,43

Tabuľka 5.2: Tabuľka hodnotení typov otázok.

Dotazník a sledovanie niekoľkých používateľov odhalilo niekoľko nedostatkov. Medzi niektoré odhalené nedostatky, ktoré boli poznamenané pri oboch skupinách, patrí nutnosť používateľa sa prihlásiť do aplikácie aj hneď po registrácii (prihlásenie sa nevykonalo automaticky) alebo okamžité odhlásenie používateľa ihneď po stlačení tlačidla odhlásiť (používatelia očakávali modálne okno pre potvrdenie svojej voľby).

Prvá skupina poukázala na možnosť zmazania všetkých možností pri vytváraní otázky, čo im dávalo pocit, že niečo pokazili. Riešením tejto situácie bolo zabránenie zmazaniu možnosti pokiaľ je v otázke ako jediná. Ďalej poukázala, že nutnosť pomenovať otvorenú možnosť v kombinovaných otázkach nie je úplne intuitívna.

Pri druhej skupine sa zistilo, že pôvodné 3 minúty neboli dostatočné množstvo času pre použitie získaného kódu. Navýšenie na 5 minút už predstavovalo dostatočný čas na jeho použitie.

Kapitola 6

Záver

Navrhnutý systém sa v práci podarilo implementovať. S jeho pomocou je možné vytvárať a distribuovať dotazníky. Podarilo sa vytvoriť aj motivačný systém, zahrňujúci získavanie tokenov, vytváranie kupónov a získavanie zľavových kódov. Bádateľ je zároveň schopný sledovať výsledky dotazníku už počas toho, ako je dotazník otvorený a nemusí čakať na jeho uzatvorenie a výsledky z dotazníku môže extrahovať do externého súboru.

Systém narozdiel od väčšiny ostatných podobných systémov umožňuje používateľom prehľadávať dostupné dotazníky a vypĺňať ich v aplikácii. Umožňuje navyše všetkým používateľom vytvárať dotazníky. Veľkým rozdielom medzi mojím systémom a ostatnými je, že každý používateľ môže pridávať aj zľavové kupóny. Tieto kupóny je potom jednoduché ako aj zakúpiť, tak aj aktivovať.

Pokračovanie práce sa môže venovať pridaniu ďalších typov otázok, ako aj pridaniu možnosti preskočiť otázku. Možné rozšírenie môže vytvoriť možnosť cielenia dotazníkov na konkrétnu skupinu potencionálnych respondentov na základe nejakej podmienky alebo sa zamerať na zlepšenie grafického rozhrania mobilnej aplikácie. Ďalšia možnosť rozšírenia je vytvoriť aplikáciu aj pre iné platformy ako iOS (napr. Android, Windows alebo Web).

Literatúra

[1] Apple. *Managing Your App's Life Cycle: Respond to system notifications when your app is in the foreground or background, and handle other significant system-related events* [online]. [cit. 19. februára 2021]. Dostupné z: https://developer.apple.com/documentation/uikit/app_and_environment/managing_your_app_s_life_cycle.

[2] Apple. The powerful programming language that is easy to learn. *Swift* [online]. [cit. 20. marca 2021]. Dostupné z: https://developer.apple.com/swift/.

[3] Debois, S. *10 Advantages And Disadvantages Of Questionnaires* [online]. [cit. 17. apríla 2021]. Dostupné z: https://surveyanyplace.com/blog/questionnaire-pros-and-cons/.

[4] Duff, V. *How to Motivate People Into Taking a Survey* [online]. [cit. 25. apríla 2021]. Dostupné z: https://smallbusiness.chron.com/motivate-people-taking-survey-75053.html.

[5] Hoffman, J. *Mastering Swift 4*. Packt Publishing Ltd, 2017. ISBN 978-1-78847-780-2.

[6] Hudson, P. *What's the difference between @ObservedObject, @State, and @EnvironmentObject?* [online]. [cit. 25. apríla 2021]. Dostupné z: https://www.hackingwithswift.com/quick-start/swiftui/whats-the-difference-between-observedobject-state-and-environmentobject.

[7] Imperva. *OSI Model* [online]. [cit. 20. apríla 2021]. Dostupné z: https://www.imperva.com/learn/application-security/osi-model/.

[8] K.Malhotra, N. *Questionnaire Desigh and Scale Development.*

[9] Lindemann, N. *35 Ways to improve your survey response rate* [online]. [cit. 25. apríla 2021]. Dostupné z: https://surveyanyplace.com/improve-survey-response-rate/#The%20right%20type%20of%20question.

[10] Malhotra, N. K. *Marketing Research An Applied Orientation.* Pearson Education, Inc., 1993. 118 s. ISBN 978-0-13-608543-0.

[11] McLeod, S. *Questionnaire: definition, examples, design and types* [online]. [cit. 17. apríla 2021]. Dostupné z: https://www.simplypsychology.org/questionnaires.html.

[12] Nelen, S. *Survey Questions: An Essential Guide to Survey Question Types* [online]. [cit. 25. apríla 2021]. Dostupné z: https://surveyanyplace.com/survey-questions-types/.

[13] Orlov, B. *IOS Architecture Patterns* [online]. [cit. 14. marca 2021]. Dostupné z: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52.

[14] QuestionPro. *The ultimate guide to great questionnaires* [online]. [cit. 25. apríla 2021]. Dostupné z: https://www.questionpro.com/blog/what-is-a-questionnaire/.

[15] Wenemark, M., Persson, A., Brage, H. N., Svensson, T. a Kristenson, M. Applying Motivation Theory to Achieve Increased Response Rates, Respondent Satisfaction and Data Quality. *Journal of Official Statistics*. 2011, zv. 27, č. 2, s. 393–414.

Príloha A

Dotazník použitý pri testovaní

Súhlasíte, že aplikácia má prívetivé používateľské rozhranie?

- Veľmi súhlasím
- Súhlasím
- Ani súhlasím, ani nesúhlasím
- Nesúhlasím
- Veľmi nesúhlasím

Páči sa Vám dizajn aplikácie?

- Áno
- Nie

Chýbali Vám nejaké tematické označenia dotazníkov? Ak áno, prosím vyplňte ktoré.

- Nie
- Áno, chýbali mi:

Chceli by ste nám ešte niečo povedať?

Príloha B

Obsah priloženého pamäťového média

Priložené pamäťové médium obsahuje nasledujúce adresáre a súbory:

- *docs/* - priečinok obsahujúci tutoriál,
- *FISEC.pdf* - textová časť práce,
- *LICENSE* - súbor obsahujúci licenciu projektu,
- *src/* - priečinok obsahujúci zdrojový kód serverovej, mobilnej a testovacej aplikácie,
- *tex/* - priečinok obsahujúci zdrojové texty textovej časti práce,
- *README.md* - popis obsahu pamäťového média vo formáte markdown.